DeepSpeech Testing Report

Accuracy and Hot-Word Feature

Project for Software Testing course in Jagiellonian University

14 January 2021

Introduction

This report contains information on black box testing process to measure usability of Mozilla's STT: **DeepSpeech** in practice. Analysis on **default English model's accuracy** with default scorer was made and **hot-words** feature was explored as it's an experimental feature without much documentation. This may serve as a suggestion whether someone would want to use certain features or current English model for their system.

Particularly, testing process helps in answering those questions:

- How accurate is DeepSpeech for accented speech?
- How accurate is DeepSpeech for speech that contains proper nouns?
- How accurate is DeepSpeech for different genders?
- How accurate is DeepSpeech for everyday, much faster, speech?
- What can be hot-words feature used for?
- Is hot-words feature bug-free?
- What can I expect from using it?
- What's the overall predicted quality of DeepSpeech default model?

Specification:

- DeepSpeech 0.9.2, default model, default scorer
- Python bindings and Python ver. 3.9.1
- Machine OS: Windows 10

Accuracy Testing

Used Method for Accuracy Metrics

The first thing that should be considered for any STT system is its accuracy or WER (Word Error Rate). The latter metric is calculated using the Levenstein's distance, where deletion, insertion and substitutions are done on words. Number of those operations is divided by reference expected transcription.

WER = LDistance/ExpectedWordsCount

Accuracy = 1 – WER

Letter Accuracy is also included in this report, which works similarly to Word Accuracy. However, this should be not taken seriously as phonetical reading of an English language word is very different to its writing. This metric may seem to be useless, but it can found use in systems, where word structure would be considered, like filtering in search engines.

Dataset for accuracy testing **250 different audio files** and divided them by categories:

- Gender (Male, Female, Other)
- Way of Speaking (Common – faster, short time between words, Lecture – slow, Emotional/Comedy – everything else, special way of speaking in other media as games or funny videos)
- Accent (True, False) – has a strong, hearable accent (other than US, may be slightly British-like)
- Has Proper Nouns (True, False) – if it does contain proper nouns (country names etc.)
- Scientific Language (True, False) – if it does contain words connected to some branch of science that normal people in everyday speech rarely use.

Then accuracy was calculated for all combinations of tags: for combinations (e.g., male, accent, common) and for broader combinations (e.g. only male)

Workflow

Input audio files were taken mostly from YouTube videos. Recording a few seconds of speech, making a human made transcription and then separating it into tag combinations was an effective technique. For converting them into correct format: 16kHz mono wav we used a *ffmpeg*. Each file had to be noise-free. Frequency and mono requirement were necessary for default DeepSpeech model for it to work correctly.

Problems of the Dataset

Usage of audio files from many sources, that wasn't meant for STT usage, could simulate a myriads of usage scenarios for DeepSpeech. This was an expensive decision, because a lot of time was spent on collecting input files. The number of input files and combinations, such as **female** and **accent** and **common** and **scientific language**, had to be reduced, as finding those audios was near an impossible. Remember that each file can't have any sounds in the background, so this also restricts our potential data.

Because it is hard to generate input data, especially if you're out of budget, testing process was restricted in terms of choosing method. Moreover, some tags were more important than others and one tag can affect the other.

Method

That's why testing process was focused on checking how **one audio feature** (in this case: **tag**) changed the **relative** accuracy of similar **tag combination**. Main classes for comparison:

- Compare pair feature: Common Male, Lecture Male, Common Female, Lecture Female (+ no accent, no science words, no proper nouns)
- Compare feature: Accent vs. No Accent
- Compare feature: Has Proper Nouns vs. Has No Proper Nouns
- Compare feature: Has Science Words vs. Has No Science Words

Warning: the summary result for all files or with broader tags should be taken with grain of salt as the number of files in one combination may affect other combinations.

If dataset had only 100 accented male, 10 non-accented male, 10 accented female and 100 non-accented female audio files and assumed that accented voice lowers the accuracy, then it would be assumed that model's accuracy for male voices is lower than for female voices.

Dataset Information and Results

Analysis

From results we can clearly see that:

- Female, unaccented, lecture speech without any proper nouns or scientific words was the most accurate. **(95.6%)**
- Male voices (+ combination as above) were less accurate, however there were more files for male speech in this combination or input files may have been just a little bit harder for a model to understand. **(94.0%)**
- **Accent heavily lowers the accuracy** (about 83.6% accuracy for lecture and **82.5%** common speech) where lecture speech of non-accented speaker is above **94.6%**
- Proper nouns reduce accuracy even more (about **10%** accuracy reduction in our data set for those that contained at least one of them), note that if we used more proper nouns then this accuracy difference could have been higher. It all depends on number of those words; however, this serves as a proof that in fact the difference is real and should be considered.

- Female common speech with no additional tags has lower accurate than male common speech (**87.3%** female, **91.4%** male).
- Scientific vocabulary may be related to drop in accuracy as for males speaking in common voice and with scientific words accuracy was: **86.4%**, while the same tag combination but without scientific words achieved **91.4%** accuracy. That gives **5%** drop.

By making a comparison to the WER given on DeepSpeech Github (7.06%), we can clearly assume that this is true for non-accented lecture speech, but if anyone were using default DeepSpeech model for accented voices or for system that accepts someone who speaks faster and doesn't make pauses for each word, they should preferably get or train their own model for specified use.

For non-accented everyday, faster speech we estimate WER to be around 11.28%.

For accented lecture speech, we estimate WER to be around 16.44%.

Also, remember that this estimation is very dataset dependent and behind "accented" hides an undefined, intuitive concept of non-USA accent.

Total WER for this 250-file dataset is **14%**. Note that this result is **based on this dataset** and should **not** be used for describing DeepSpeech default English model's WER.

Experience and Tips for Better Results

The best results for our method would be to make a custom recordings of text reading by different speakers. Eventually, change some of the words of this text to match scientific vocabulary and/or proper nouns. This requires a higher budget but would give a very accurate result.

Time and Resources

Collecting audio files from YouTube using Audacity took about 12 hours in total. Human made transcriptions and tagging took another 12 hours. Creating test script took about 16 hours of work.

Hot-words Exploratory Testing

Why Exploration?

Newly added feature was something that needed to be tested, even though there wasn't much known about it. Users should know of possibilities, dangers and practices for this addition. Because of unknown, exploratory testing was useful as forcing different behaviors gave a good outlook on how this can be utilized by others.

Tested Cases of Potential Usage

Those cases are universal and describe a scenario of a behavior while using hot-words feature:

- **UWTD** - **Undetected Word to Detected** by using hot-words: (e.g.: "This is a grate" detected as "This is great", should transcribe as "This is a grate" if we choose "grate" as hot-word with some positive priority). Usage for calibrating accuracy.
- **DWAU** - **Detected Word as Undetected,** we want to eliminate chosen words from audio transcription, by giving them negative priority, in this case we want to make sure that hot-word will not be present in output transcription. (eg.: "Behold the gold as gold is just a mold", hot-word "mold"=-24.0). Usage for reducing number of words even if they are detected correctly (swear words etc.)
- **NHS** - **No Hot-words Similarities** in a sentence should not affect our transcription much (e.g.: "My parents found a diamond ring" transcription will not be changed if we set up hot-word: "item" with additional priority 15.5). Usage in different case, but for audio that will be ignored by those cases.
- **AHS** – **All Hot-word Similarities**, meaning in sentence there will be words with smaller and bigger similarities. We must check how system responds to our input. (eg. "being the biggest is bigger than being big", bigger = 20.5)

UWTD

DeepSpeech received an audio file that had "this is a grate". It was transcribed as "the great". By adding hot-word "grate" with different priorities this occurred:

Below about 20.0 priority, "grate" was still undetected.

From 20.0 to 28.0 whole audio got transcribed as: "the grate "(space after grate)

Above 28.0 the transcription: "the grate s s s s".

To check whether this was caused by the word being at the end additional test was made.

“these mails are yours” transcribed as “these males are yours”, so mails was added a priority.

```
    ['mails'] = (-5.0,) :: [the males are you]
…
    ['mails'] = (1.0,) :: [these mails are you]
…
    ['mails'] = (7.0,) :: [these mails are you]
    ['mails'] = (8.0,) :: [these mails are his as]
    ['mails'] = (9.0,) :: [these mails are his as]
    ['mails'] = (10.0,) :: [these mails are you]
    ['mails'] = (11.0,) :: [these mails are you]
    ['mails'] = (12.0,) :: [this mails a r t h you]
    ['mails'] = (13.0,) :: [this mails a r t h you]
    ['mails'] = (14.0,) :: [his mails a c a t you]
    ['mails'] = (15.0,) :: [his mails a car i e you]
    ['mails'] = (16.0,) :: [his mails a car t h you]
    ['mails'] = (17.0,) :: [his mails a c a r you]
    ['mails'] = (18.0,) :: [his mails a c a r you]
    ['mails'] = (19.0,) :: [his mails a c a r you]
    ['mails'] = (20.0,) :: [his mails a c a r e you]
```

The bug where next word is being changed into a letter sequence is still prevalent. Because of that, this does mean that for accuracy calibration it is rather hard to find a middle ground. Also, notice that other words are affected by the hot-words in very unregular way.

This bug was noticed here: https://discourse.mozilla.org/t/new-0-9-alphas-to-test-new-features/68134/4

DWAU

```
['another'] = (-75.0,) :: [and then i read an other one and nother one and an other one]
…
['another'] = (-16.0,) :: [and then i read an other one and nother one and an other one]
['another'] = (-15.0,) :: [and then i read an other one and nother one and another one]
…
['another'] = (-4.0,) :: [and then i read an other one and another one and another one]
['another'] = (-3.0,) :: [and then i read an other one and another one and another one]
['another'] = (-2.0,) :: [and then i read another one and another one and another one]
['another'] = (-1.0,) :: [and then i read another one and another one and another one]
```

['another'] = (0.0,) :: [and then i read another one and another one and another one]

The 0.0 prio transcription: "and then i read another one and another one and another one" was a correctly described transcription. We can see that for trying to reduce the occurrence of a word we must be wary of the word splitting. Next test, more fitting for **AHS** case than from **DWAU**, was made to include sub words, for example:

['another', 'nother', 'other'] = (-25.0, -25.0, -25.0) :: [and then i read an ther one and anither one and anither one]

It keeps on splitting in eating the letters because it really hards to match a correct word. This proves that un-detecting the words, may work better than detecting a word as bug that happened previously doesn't occur.

NHS

By using the same audio file as previously, the 'book' hot-word that is not in the transcription nor in the audio file as occurring word. No changes in output transcription were found.

Then 'america' was added. Even the cartesian product of positive, negative priorities of both hot-words didn't make a change. This works as we predicted.

AHS

For input:

*"why are they sad and glad and **bad** i do not know go ask your dad",*

transcription was:

*"why are they sad and glad and **that** i do not know go ask your dad"*

Adding a hot-word "bad" with positive priority forced it to transcribe correctly, but the more priority we gave the more other similar words were changing into "bad".

['bad'] = (0.0,) :: [why are they sad and glad and that i do not know go ask your dad]

['bad'] = (0.5,) :: [why are they sad and glad and that i do not know go ask your dad]

['bad'] = (1.0,) :: [why are they sad and glad and bad i do not know go ask your dad]

['bad'] = (1.5,) :: [why are they sad and glad and bad i do not know go ask your dad]

['bad'] = (2.0,) :: [why are they sad and glad and bad i do not know go ask your dad]

['bad'] = (2.5,) :: [why are they bad and glad and bad i do not know go ask your dad]

['bad'] = (3.0,) :: [why are they bad and glad and bad i do not know go ask your dad]

['bad'] = (3.5,) :: [why are they bad and glad and bad i do not know go ask your dad]

['bad'] = (4.0,) :: [why are they bad and glad and bad i do not know go ask your dad]

['bad'] = (4.5,) :: [why are they bad and glad and bad i do not know go ask your dad]

['bad'] = (5.0,) :: [why are they bad and bad and bad i do not know go ask your dad]

For higher priorities:

['bad'] = (95.0,) :: [why are they bad e d e a d glad and bad i i d i do not know go ask your bad e e e]

Adding more hot-words with similar letter structure doesn’t seem to change much in those words. Scorer just changes it’s guess according to priority. More unclear speech can generate more unclear guess when priorities are added. One shouldn’t add words of similar structure as hot-words as predicting the output is very unintuitive.

Practical Analysis and Other Findings

- Adding hot-word that has a space, like: “another one” doesn’t change behavior. Probably because it doesn’t appear in word detection mechanism and is not modified.
- Use hot-words if you need to detect one word and you can ignore everything else that comes after that word, because of letter splitting bug. Example: “okay google”.
- You can use negative priority for words no to occur in the output, but be careful of this word to appear as a splitted one: “another” -> “an other” or as a word of similar sound: “gold” -> ”god”.
- Usage hot-words for calibrating accuracy is not the perfect one but add a very small priority and it could work.
- Nonexistent words as hot-words or words that share no similarity to the given hot-word cause no change if the audio doesn’t include the sound of that word.
- No software related errors caused by **adding, removing, clearing** hot-words feature were detected.
- It’s hard to predict use cases, so more detailed testing should be made for other uses.

Time and Resources

Each testing case took about an hour of work. Audio files from accuracy testing were useful, as it reduced needed to record/create inputs.

After Action Report

Testing process on accuracy was mostly restricted by our input data. More detailed testing on different tags combinations were planned. However, miscalculations and initial optimism hide the truth of ad hoc collecting data as a laborious task. This was done in good faith, as diversity of audio files that are not used in STT training could prove its' practical value for other developers.

It was also planned to do accuracy testing on audio files with different noise levels or background sounds, but shortage of time made this impossible. Additionally, tests on the most common used microphones could be made. Expensive, but could produce an interesting result.

Testing process took about 45 hours of proper work and another 20 hours of research and additional tasks that weren't included previously such as writing this very report. This time was split on two team members.

Assuring the quality of Mozilla's DeepSpeech was a time-consuming process but we hope that the conclusions of our work would be helpful for everyone who want to use this STT system in their work.

Verification

All used files and scripts are here. For script to work correctly: add folder named 'workspace'. We can't add audio files as those may be licensed.

Dominik Zimny and Matusz Zdon

for *The Mozilla Foundation*