Data Augmentation for Natural Language Text Datasets

by

Anooshik Vartanian

Bachelor of Software Engineering, Shahid Beheshti University, 2016

A thesis submitted in partial satisfaction of

the requirements for the degree of

Master of Science

in

Computer & Information Science

in the

COLLEGE OF SCIENCE AND ENGINEERING

of the

AMERICAN UNIVERSITY OF ARMENIA

Supervisor: Adam Mathias Bittlingmayer

Signature: __________________ Date:_____________

Committee Member: ____________________
Signature: __________________ Date:_____________

Committee Member: ____________________
Signature: __________________ Date:_____________

Committee Member: ____________________
Signature: __________________ Date:_____________

This study is dedicated to my parents; my mother, Shakeh and my father, Vahik.

I appreciate all the sacrifices you made for me to get to this point in my life.

This paper would not have been possible without your encouragement, support and endless love.

Acknowledgement:

I would like to thank, first and foremost, my supervisor, **Adam Mathias Bittlingmayer**, for his trust and motivation throughout developing and writing the paper. I appreciate all the times he has provided professional guidance during this study.

My gratitude extends to my wonderful mentors, colleagues and friends in **Informatics Solution** for providing me with supporting working environment and having shown deep understanding in the chaotic life of a Master's degree student!

Abstract:

In this thesis, I explored the idea of using data augmentation for text datasets through developing a python application. Chapter one explains the problem statement and what I planned to achieve. Chapter two provides background information on definitions and tools used during the study. Chapter three focuses on the structure and contents of the python program as well as the features it provides. Chapter four is about the result of using the application on three datasets. Chapter five presents the summary and conclusion to the thesis, as well as some suggestions and improvements for future.

Keywords: Perturbation, Python, Natural Language Processing

Licenses for Software and Content

Software Copyright License

Content Copyright License

[illegible]

Table of Contents:

Chapter One: Thesis Overview

Introduction

This chapter explores the general overview of the project, both the importance and necessity of implementing the project, the steps involved in choosing a project to implement and producing quality software. At first, by stating the problem, I will describe the question this thesis has attempted to solve, and then explain the objectives of the implementation and the assumptions contained in the outline of the plan.

Problem Statement

One of the problems data scientists face in text datasets is getting clean data to analyze and extract meaningful materials from it. The main issue here is that many categories in Natural Language Processing (NLP) come from texts that are gathered from social media sites or other sources that are not necessarily in the conventional language format. If the data has some errors, for instance, it is misspelled, then that row may become useless and lead to wrong results. The data, subsequently, must either be removed from the dataset or the data scientist can attempt to correct and clean that information. The data has to be removed before the training phase but not on live queries, whereas the data can be fixed both during the test phase and on live queries.
Both methods, eliminating the row or attempting to detect or fix the error have their challenges. Removing the row might not be the best solution when the dataset is small and would ignore data that could be useful. Attempts to fix and clean the data also bring out a new set of problems. The error should be detected using algorithms and then the data should either be fixed manually or using algorithms.

When dealing with data in multiple languages, as is often the case in building a global model, cleaning the data will take valuable resources.
Both options mentioned above are valid and are used in many projects that work with text data. This study attempts to use another approach, in which case the data, though messy, is kept and used to improve the overall accuracy of the final model. The approach uses data augmentation, which is a known technique. It uses machine learning to learn robustness to specific types of noise and prevent overfitting, i.e., improve out-of-domain results.
Augmentation approach is also used in case of relatively small datasets, where overfitting can often occur. By generating new data based on existing dataset, the model would become larger and solve the general overfitting problem. A simple example of what the data augmentation wishes to achieve is shown in the sequence of sentences below.

Autumn has started, but the leaves in NYC haven't changed color yet. (Original)
Autumn's *started but the leaves in* ***N.Y.C*** *haven't changed color yet.* (Substitute punctuation)
The leaves in NYC haven't changed color yet, ***but Autumn has started****.* (Reordering sentence)
Autumn has started but the ***leafs*** *in NYC have not changed color yet.*
Fall *has started, but the leaves in NYC haven't changed* **colour** *yet. (Substitute Words from American to British)*

This class of techniques stands in contrast to lossy pre-processing that normalizes or filters data, i.e., reduces noise or data, and which must be applied to real data at runtime exactly as it is applied to the training data.

Goals and Objectives

The primary goal of this study is to develop an open-source production-ready

Python library that will augment a set of sentences to include many types of reasonably realistic mutations. The library will generate a new augmented dataset that can be used to train more accurate models. By evaluating the new dataset on benchmarks for various natural language tasks, the study can understand the effectiveness of different mutations and find out under which parameters it performs better on in-domain and out-of-domain data.

The objectives of this study are:

- Determining the parameters for mutations that would generate better results for most problems.
- Showing that using text data augmentation can yield significant increases in accuracy of the model for common tasks.
- Providing an easy-to-use library for machine learning projects.

Chapter Two: Theoretical Background

Machine Learning

Machine Learning systems learn how to combine input from model to form an algorithm that could, in the long run, produce useful predictions on data that it had never encountered before. [1]
The algorithms devised for Machine Learning are typically categorized into the following categories:

- Supervised learning
- Unsupervised learning
- Semi-Supervised learning
- Reinforcement learning

This study focuses on supervised learning. In supervised learning, a dataset called training dataset acts as a teacher. By studying the training data, it will lead to learning a general rule that maps data to labels. A label or a class is what the goal of prediction is. The label could be the stock price of Google next month, the kind of animal shown in a picture, the language of a sentence, the meaning of an audio clip.
Supervised learning further splits into two broad categories: Regression and Classification.
Classification assigns a label to the model from a finite set of classes to an observation. That is, responses are categorical variables and the machine must provide a model for distinguishing between two or more discrete labels. For example, a natural language processing classification model could determine whether an input sentence was in Armenian, English, or Persian.
After training the model on the training dataset, the model is checked on the validation or testing datasets.
Accuracy - is the measure of the effectiveness of the machine learning model. Accuracy is calculated by dividing the number of labels guessed correctly to all the

rows in the dataset.

Accuracy = Number of correct predictions / Total number of predictions

However, it does not mean that the model with the highest accuracy has the best performance.
True Positives (TP) - This refers to the correctly predicted positive values i.e. the value of both the actual class and predicted class is yes.
True Negatives (TN) - These are the correctly predicted negative values i.e. the value of both actual class and predicted class is no.
False Positives (FP) - It indicates values where the actual class is no but the predicted class is yes.
False Negatives (FN) - It indicates values where the actual class is yes but predicted class in no.

		Actual Class	
Predicted Class		Positive	Negative
	Yes	True Positive (TP)	False Positive (FP)
	No	False Negative (FN)	True Negative (TN)

Based on these values there are two more measures to calculate the performance: precision and recall.
Precision is the ratio of correctly predicted positive values to the total predicted positive values. This metric highlights the correct positive predictions out of all the positive predictions. High precision indicates low false positive rate.
Precision attempts to answer the following question:
What proportion of positive identifications was correctly calculated?
The formula for precision is:

Precision = TP / (TP + FP)

Recall is the number of correct positive results divided by the number of all relevant samples. Recall attempts to answer the following question:
What proportion of actual positives was identified correctly?
The formula for recall is:

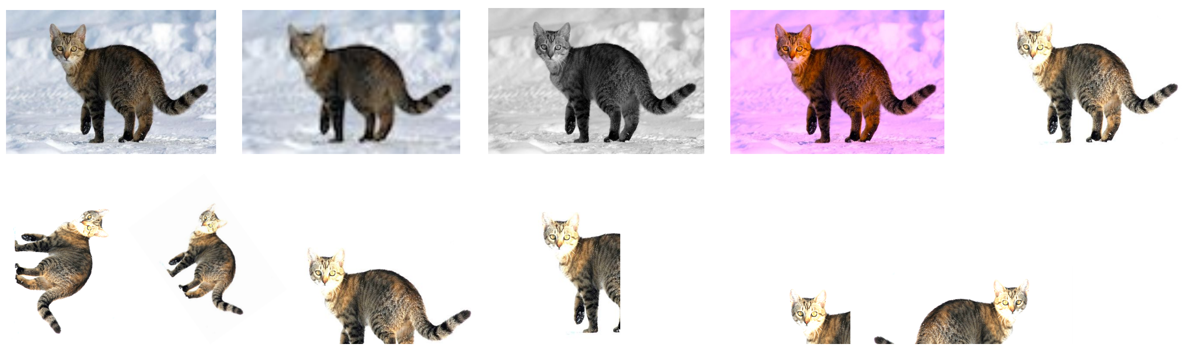
$$\text{Recall} = TP / (TP + FN)$$

The F1 score or F-score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1(perfect precision and recall) and worst at 0.

$$\text{F-Score} = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

Data Augmentation

Data augmentation is a class of techniques in machine learning to 1) learn robustness to specific types of noise 2) prevent overfitting i.e. improve out-of-domain results.
This class of techniques stands in contrast to lossy pre-processing that normalizes or filters data i.e. *reduces* noise or data, and which must be applied to real data at runtime exactly as it was applied to the training data.
A typical approach for supervised tasks is to preserve the annotation while mutating the data in some way. This is standard practice and is effective for tasks that use image data [2] and has been tried for audio data as well [3], but little is known about how it can be used for text data.
For example, for an image labelled **cat**, the dataset can be augmented with images generated with combinations of exposure, saturation removal, over-saturation, flipping, rotation, cropping, stretching and so on and so forth.



Natural language text data is more challenging to mutate realistically because it is not in a continuous space - even small mutations can generate invalid data that would never occur in real datasets.

fastText

fastText is an open-source library designed to help build scalable solutions for text representations and will allow users to learn word representations and text classifications. [4] Text classification includes tagging each document in the text with a particular class. Sentiment analysis and email classification are classic examples of text classification.
fastText has implemented the following options to train and then test the model. The following commands are all that needed to be familiar with for this study. [5]
supervised- train a supervised classifier
test- evaluate a supervised classifier
lr- learning rate, default value is 0.05. Learning rate determines how fast weights given to algorithm of determining labels change.
epoch- number of epochs, default value is 5. The number of epochs is the number of times each example is seen.
dim- size of word vectors, default value is 100. The dim value greatly affects the size of the model and consequently affects the performance.

Chapter Three: Methods

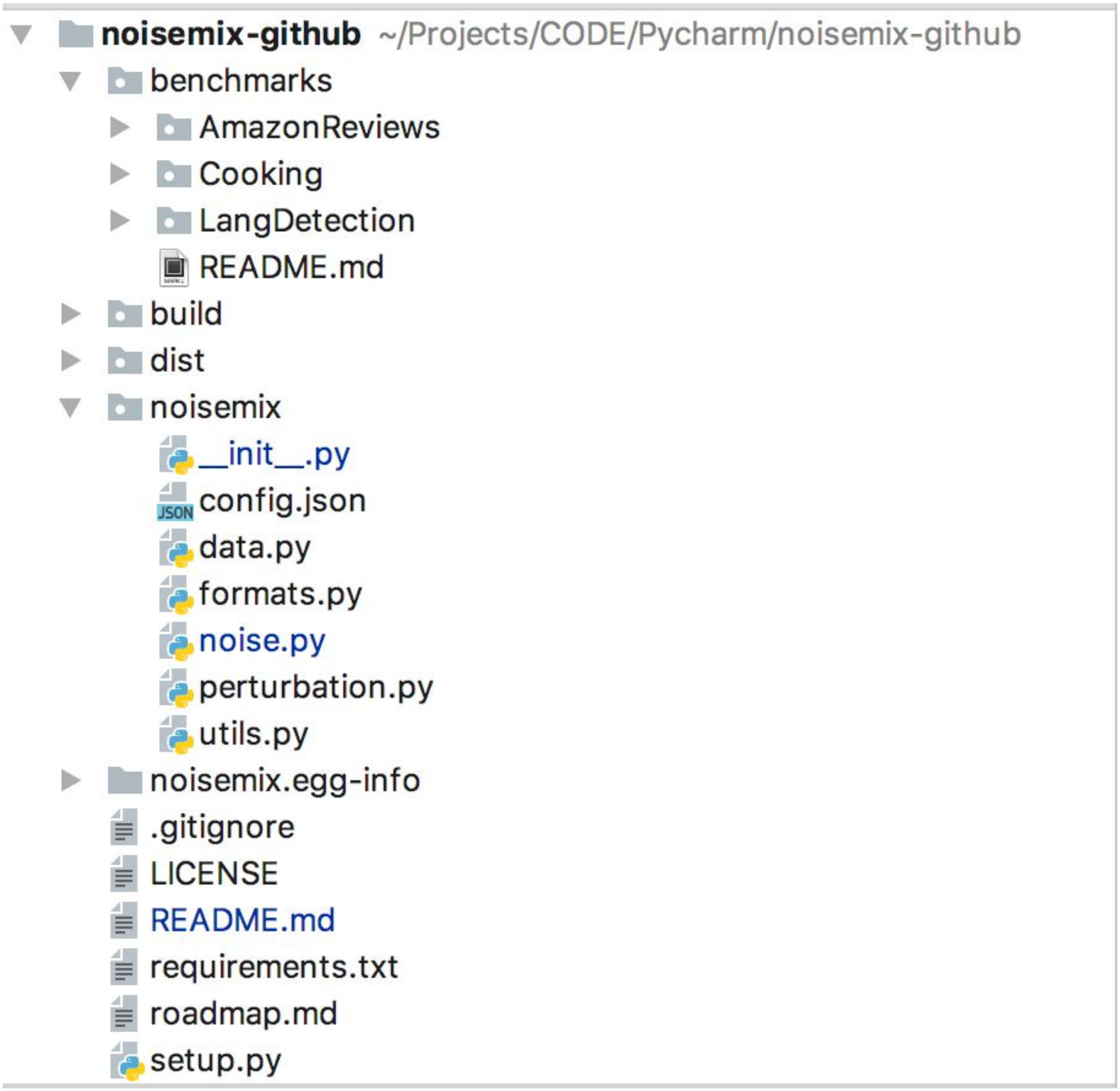
Introduction

This chapter provides the technical development and structure of the program by explaining project files and their content. Then the functions provided by the program and the variables used to customize them are explained, providing the examples.

Structure

The program developed for this thesis is written in Python language and aims to provide a tool for developers and data scientists to apply data augmentation to their datasets.

First version of the program contained a simple input and output flow. The input was the path to the text file, the program would read the file line by line. Each line would then be divided as a list of words. For each word, there would be a random function determining what perturbation to use. For example, if the random was from range [0, 0.2) it would add a random letter from alphabet to that word. The output would be the collection of all the perturbations in the form of another text file. The project was developed in a single .py file.
As the program evolved, the flow changed and the classes expanded. The current structure of the project is visualized in the picture below:



```
▼ noisemix-github  ~/Projects/CODE/Pycharm/noisemix-github
  ▼ benchmarks
    ▶ AmazonReviews
    ▶ Cooking
    ▶ LangDetection
      README.md
  ▶ build
  ▶ dist
  ▼ noisemix
      __init__.py
      config.json
      data.py
      formats.py
      noise.py
      perturbation.py
      utils.py
  ▶ noisemix.egg-info
    .gitignore
    LICENSE
    README.md
    requirements.txt
    roadmap.md
    setup.py
```

The program consists of one main package, noisemix. The contents of the package are explained below:

__init__.py contains the main function and is called with various variables, most importantly the location of the input file and the number of versions it should produce for each row. It then divides the sentences to words and calls word perturbation function from noise.py file. Then, it joins the words and calls the sentence perturbation function. The result is then written to a file in the same directory as input file. The program also has the ability to print the result in the

command line for easy viewing.
config.json includes the list of constants that can be set and changed by the user of application. The file is read and the values applied when initializing noise.py file. Further explanation of this file is provided in Variables section.
data.py includes a list of word, character and keyboard transformations. Now the only supported language is English.
word_trasnformations contains a list of commonly mistaken, or otherwise interchangeable words in English. For example it includes [“there”, “their”, “they’re”] as well as changes in dialect [“grey”, “gray”] or other interchangeable words such as [“2”, “two”].
char_transformations provides a similar functionality for punctuations, which visually look the same, but has different encoding. The table below shows different unicodes for the dot character. [6]

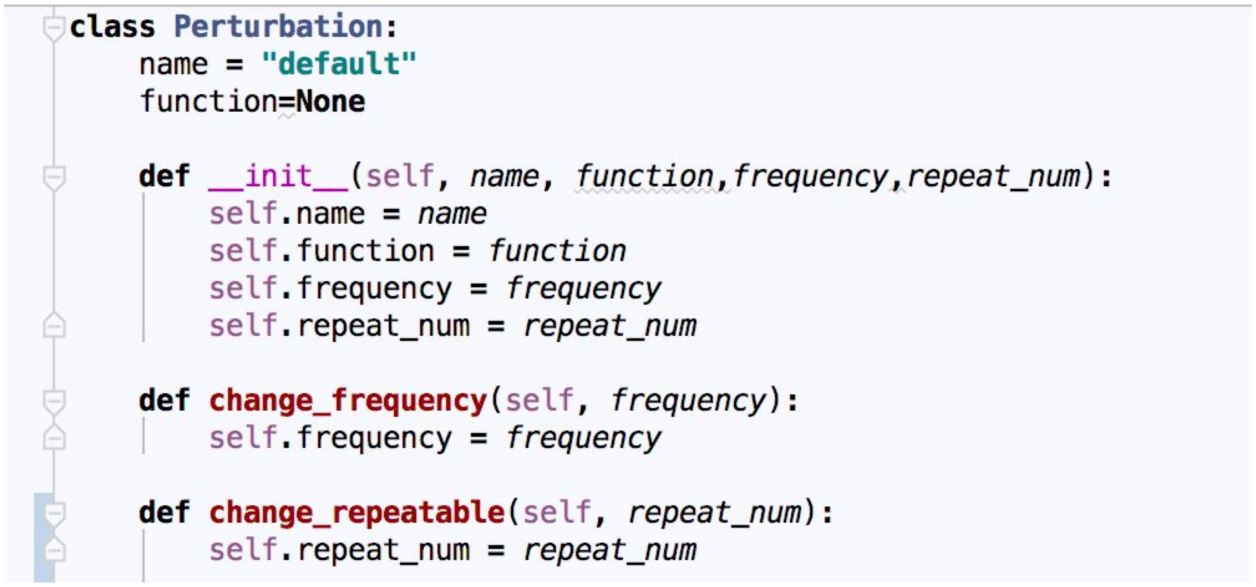
Character Unicode	Character Name	Character
U+00B7	MIDDLE DOT	·
U+002E	FULL STOP	.
U+2024	ONE DOT LEADER	․

keyboard_transformations takes the layout for keyboard and is used for typo_qwerty perturbation, which simulates how a misspelling can be produced if the user types the neighbor key on the keyboard. Now, the supported keyboard layout is QWERTY layout, since it is commonly used on standard keyboard layouts on English-language computers and mobile devices.
It is expected, that this collection would be expanded as support for other languages is added.
formats.py There are some words or expressions in each rows that should not be augmented. For example, when using fastText data, each row has a prefix marked __label__ that determines the classification label. In this python file, it is assured that the label remains unchanged. __init__.py calls the function for this file before adding the row to dataset. Now the only supported format is fastText.
noise.py contains two main functions randnoise and sentence_noise. Upon first

load it initializes different perturbations. Both functions essentially perform the same action for word and sentence inputs, respectively. They take a random collection of available perturbations and randomly choose whether the perturbation can be applied to string or sentence.

perturbations.py For each member of the class contained the function name and redirected to the respective function. It includes a member which indicated probability of getting chosen, called frequency, and the number of times the perturbation could get repeated in a sentence, repeated_num.

The image below shows the structure of the class, along with the ability to set different values for perturbation frequency and number of times they could be changed runtime.



```python
class Perturbation:
    name = "default"
    function=None

    def __init__(self, name, function,frequency,repeat_num):
        self.name = name
        self.function = function
        self.frequency = frequency
        self.repeat_num = repeat_num

    def change_frequency(self, frequency):
        self.frequency = frequency

    def change_repeatable(self, repeat_num):
        self.repeat_num = repeat_num
```

Functions

A list of the perturbation with description of each of their functionalities is explained below.

add_letter

Adds a random letter from the alphabet to a random position of the word.

Example: track -> traock (addition of o to fourth position)

repeat_letter
Chooses a random letter from the word and duplicates it.
Example: best -> beest (adding e in second position)

Remove_letter
Chooses a random letter from the word and removes it.
Example: soundtrack -> sountrack (removing n in fifth position)

lowercase
Changes the word to lowercase.
Example: Mind -> mind

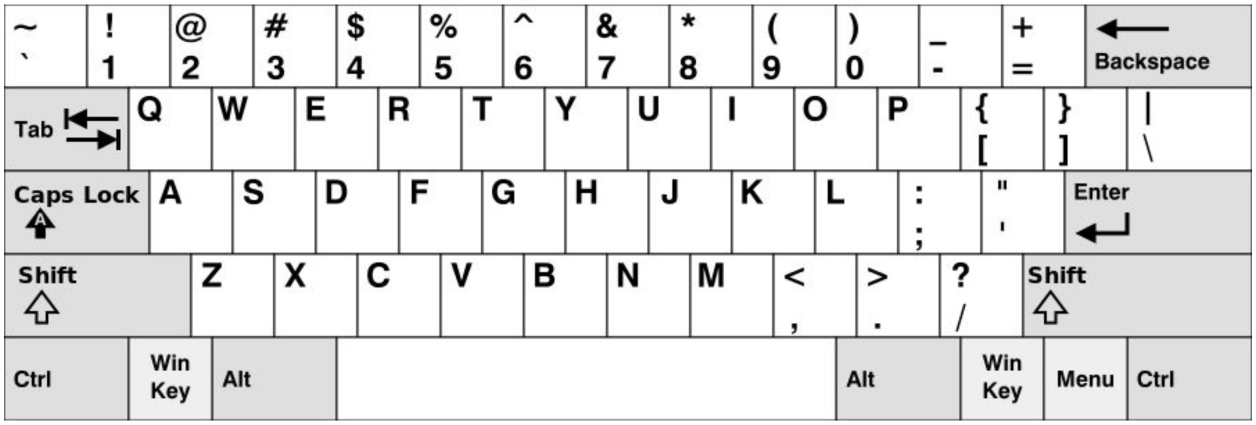
remove_punct
Removes any punctuation characters from the word.
Example: beautiful! -> beautiful

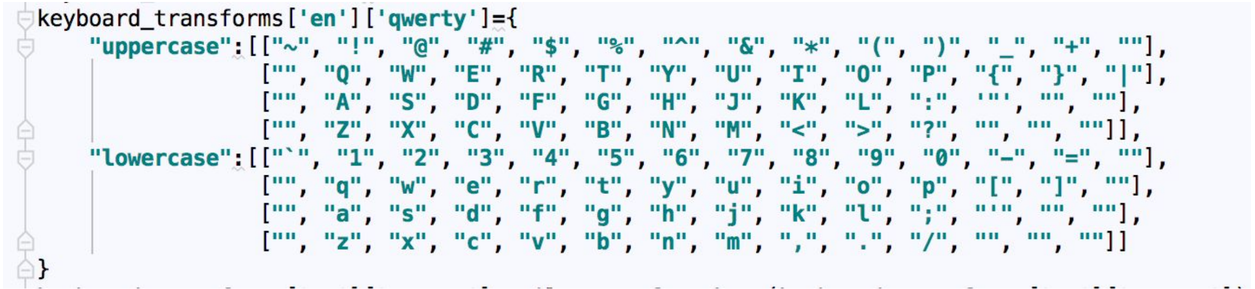
word_swap
The entire word is replaced by a similar word in the following cases: the words are common mistakes (they're-> their); there are differences in dialects (color -> colour); they can be used interchangeably (and->&).
Example: percent? -> %?

char_swap
Replaces the character with similar looking character but with different encoding.
Example: game. -> game· (The dot in fifth position (unicode U+00B7) is replaced with similar looking dot (unicode U+002E)

flip_letters
Chooses two random letters of the word and switches their places.
Example: would -> wolud (switching l and u in second and fourth position)

typo_qwerty
Let's start by examining a common qwerty layout keyboard.



If we imagine them as two dimensional array for uppercase and lowercase letters, they would look like the image below:


```
keyboard_transforms['en']['qwerty']={
    "uppercase":[["~", "!", "@", "#", "$", "%", "^", "&", "*", "(", ")", "_", "+", ""],
                 ["", "Q", "W", "E", "R", "T", "Y", "U", "I", "O", "P", "{", "}", "|"],
                 ["", "A", "S", "D", "F", "G", "H", "J", "K", "L", ":", '"', "", ""],
                 ["", "Z", "X", "C", "V", "B", "N", "M", "<", ">", "?", "", "", ""]],
    "lowercase":[["`", "1", "2", "3", "4", "5", "6", "7", "8", "9", "0", "-", "=", ""],
                 ["", "q", "w", "e", "r", "t", "y", "u", "i", "o", "p", "[", "]", ""],
                 ["", "a", "s", "d", "f", "g", "h", "j", "k", "l", ";", "'", "", ""],
                 ["", "z", "x", "c", "v", "b", "n", "m", ",", ".", "/", "", "", ""]]
}
```

We can calculate the distance between two keys by a simple distance formula. In which x1,y1 is the position of first key in the two dimensional array and x2,y2 is the second key.

dist = sqrt((x2-x1)^2 + (y2-y1)^2).

When the program first starts, we collect a list of all key distances from each other in key-pair values. For example, the list for 'f' would be [('r', 1), ('d', 1), ('g', 1), ('v', 1), ('4', 2), ('e', 2), ('t', 2), ('s', 2), ('h', 2), ('c', 2), ('b', 2), ('2', 3), ('3', 3), ('5', 3), ('6', 3), ('w', 3), ('y', 3), ('a', 3), ('j', 3), ('x', 3), ('n', 3), ('1', 4), ('7', 4), ('q', 4), ('u', 4), ('k', 4), ('z', 4), ('m', 4), ('`', 5), ('8', 5), ('i', 5), ('l', 5), (',', 5), ('9', 6), ('o', 6), (';', 6), ('.', 6), ('0', 7), ('p', 7), ("'", 7), ('/', 7), ('-', 8), ('[', 8), ('=', 9), (']', 9), ('', 10)]

The function typo_qwerty takes the closest neighbors by some margin. The default value is set at two. Therefore, neighbors with values 1 and 2 are considered in the function to replace the letter. If we consider the example for 'f' it would be [('r', 1), ('d', 1), ('g', 1), ('v', 1), ('4', 2), ('e', 2), ('t', 2), ('s', 2), ('h', 2), ('c', 2), ('b', 2)]

Example: recommend -> revommend (c in third position is replaced by neighbor v)

remove_space

Removes the space between a random word and the next word in the sentence.
Example: I would recommend this game to everyone. -> I would recommendthis game to everyone.

flip_words
Swaps a random word with the next word in the sentence.
Example: This soundtrack was beautiful! -> This was soundtrack beautiful!

Variables

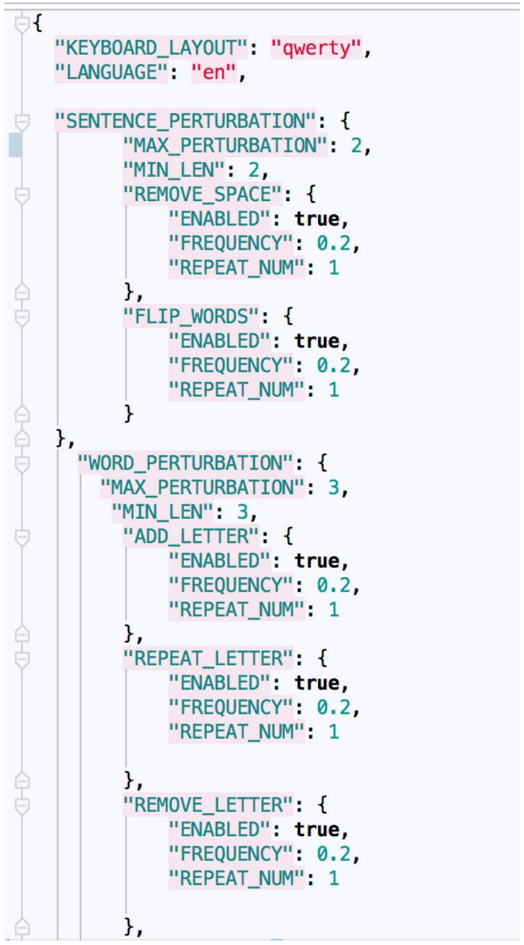
One of the ways to make the program versatile and easy to use is to avoid hard-coded variables and instead, give the users the ability to modify them. I have decided to do that by providing basic and commonly used settings through passing variables via command line. The more detailed customization can be done via editing the values in a configuration file, **config.json**. The advantage of using external file is that this way it loads the configuration values defined in external file, not the built-in data structures. It is also more general way since it is not considered to be part of code, so the users would not need to build or perform any other action. Both types of variables and their usages are listed below:

Command line:

These variables can be configured when running the main (__init__.py file)
path , the path to input data file**.**
format, if the dataset has labels or other variables that need to be excluded from perturbation. It currently supports fastText.
versions, how many versions to generate per line, default value is 1.
print, allows the user to print the output instead of writing to file, default value is 0.

Config.json:

The image below displays part of the config.json file, which is read as the program runs for the first time.



```
{
  "KEYBOARD_LAYOUT": "qwerty",
  "LANGUAGE": "en",

  "SENTENCE_PERTURBATION": {
        "MAX_PERTURBATION": 2,
        "MIN_LEN": 2,
        "REMOVE_SPACE": {
            "ENABLED": true,
            "FREQUENCY": 0.2,
            "REPEAT_NUM": 1
        },
        "FLIP_WORDS": {
            "ENABLED": true,
            "FREQUENCY": 0.2,
            "REPEAT_NUM": 1
        }
  },
    "WORD_PERTURBATION": {
      "MAX_PERTURBATION": 3,
       "MIN_LEN": 3,
        "ADD_LETTER": {
            "ENABLED": true,
            "FREQUENCY": 0.2,
            "REPEAT_NUM": 1
        },
        "REPEAT_LETTER": {
            "ENABLED": true,
            "FREQUENCY": 0.2,
            "REPEAT_NUM": 1

        },
        "REMOVE_LETTER": {
            "ENABLED": true,
            "FREQUENCY": 0.2,
            "REPEAT_NUM": 1

        },
```

KEYBOARD_LAYOUT, is used for typo_qwerty function and is a global variable.
LANGUAGE, the default language for dataset.
SENTENCE_PERTURBATION, consists of members MAX_PERTURBATION and MIN_LEN which portrays the maximum number of perturbations to be used per line and the number of words the minimum sentence length must include to perturbate the sentence.
Then, each perturbation function is configured with "ENABLED", "FREQUENCY" and "REPEAT_NUM" variables.
The same settings are available for WORD_PERTURBATION and its respective functions. The user can change the aggressiveness of noises and decide whether to include some augmentations from the following file.

Data

I tested the algorithm on datasets provided by three sources. Both data were divided to test and train models.
The first dataset consists of customer reviews [7] as input text and star ratings as output labels for sentiment analysis. The negative and positive reviews are labeled as seen by examples below.

__label__1 The Worst!: A complete waste of time. Typographical errors, poor grammar, and a totally pathetic plot add up to absolutely nothing. I'm embarrassed for this author and very disappointed I actually paid for this book.

The second dataset contains examples of questions from the cooking section of Stackexchange [8], and their associated tags. The input is stackexchange questions about cooking and the output is the possible tags, such as pot, bowl or baking. In this dataset, the input can have multiple labels.
__label__equipment __label__knives What should I look for in a good, multi-purpose chef's knife?

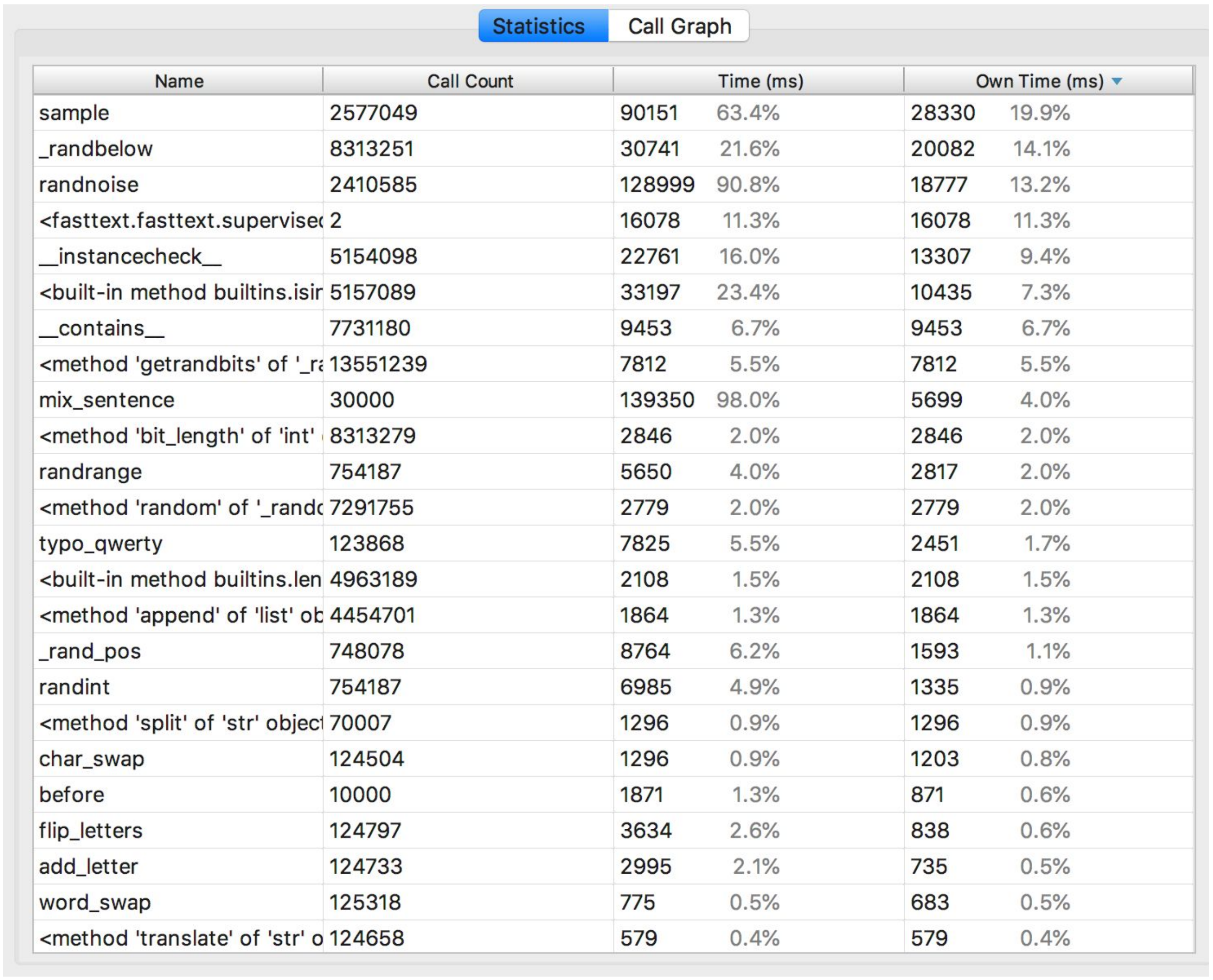
The third dataset contains sentences from the Tatoeba website [9], which is a collection of sentences and translations. The input are the sentences and output labels are indicating the language is the result.
__label__fra Le champ a désespérément besoin d'eau.
__label__eng Tom stopped suddenly.

These three datasets' results are further discussed on Chapter Five.

Performance

The program will be more useful for other developers, if it runs in a reasonably quick time and generates new version of dataset.
Profiler [10] measures the duration and number of times each function is called. I

was able to detect from the image below the functions that took up most of the time was the random function. The result is from running the program to produce three versions of train set of 10,000 rows of amazon review data sentiment analysis code.



Name	Call Count	Time (ms)		Own Time (ms)	
sample	2577049	90151	63.4%	28330	19.9%
_randbelow	8313251	30741	21.6%	20082	14.1%
randnoise	2410585	128999	90.8%	18777	13.2%
<fasttext.fasttext.supervise	2	16078	11.3%	16078	11.3%
__instancecheck__	5154098	22761	16.0%	13307	9.4%
<built-in method builtins.isir	5157089	33197	23.4%	10435	7.3%
__contains__	7731180	9453	6.7%	9453	6.7%
<method 'getrandbits' of '_ra	13551239	7812	5.5%	7812	5.5%
mix_sentence	30000	139350	98.0%	5699	4.0%
<method 'bit_length' of 'int'	8313279	2846	2.0%	2846	2.0%
randrange	754187	5650	4.0%	2817	2.0%
<method 'random' of '_rando	7291755	2779	2.0%	2779	2.0%
typo_qwerty	123868	7825	5.5%	2451	1.7%
<built-in method builtins.len	4963189	2108	1.5%	2108	1.5%
<method 'append' of 'list' ob	4454701	1864	1.3%	1864	1.3%
_rand_pos	748078	8764	6.2%	1593	1.1%
randint	754187	6985	4.9%	1335	0.9%
<method 'split' of 'str' object	70007	1296	0.9%	1296	0.9%
char_swap	124504	1296	0.9%	1203	0.8%
before	10000	1871	1.3%	871	0.6%
flip_letters	124797	3634	2.6%	838	0.6%
add_letter	124733	2995	2.1%	735	0.5%
word_swap	125318	775	0.5%	683	0.5%
<method 'translate' of 'str' o	124658	579	0.4%	579	0.4%

Sample which appears as the front row of the table is part of the random package in python. By examining the code, it was obvious that a lot of perturbation functions used random.sample() function to get a random position or letter to apply the perturbation to. Then, I decided to write a custom random function and find more efficient functions to optimize the code.

Packaging

The final step of the development was to package and distribute the Python project [11]. The Python Package Index, abbreviated as PyPI is the official third-party software repository for Python. Following the instruction, the package is set on the following link[1] and can be installed via pip install noisemix command.

I have also uploaded the program on github, so that other developers can view and contribute to the source code. The link to the github directory is located on https://github.com/noisemix/noisemix

1 http://pypi.python.org/pypi/noisemix

Chapter Four: Results

Introduction

Chapter four explores the results from three datasets. The datasets have been benchmarked before in regards to their performance. Now I will include how augmenting the new dataset will affect the performance. For each dataset, I have mentioned how many rows the dataset consisted of, and what the original precision was. Then I will try the application with some different values for parameters (version, frequency, whether to include the original data in the mix). The results are rounded up to four decimal point precision. It is important to note that the default value for word and sentence perturbation frequency is two and the default number of times each perturbation is repeated per word or sentence (repeat_num) is one.

Amazon Reviews

Overall, the data consisted of 4,00,000 rows[7]. I took ten percent for test dataset (400,000) and the rest was used for training data. The python code to run is displayed below.

```
fasttext supervised -input train.ft.txt -output model_amzn
```

The original precision for different row size of training data is show below.

	Precision
10,000 rows	0.8151
20,000 rows	0.8528
30,000 rows	0.8641
40,000 rows	0.8708
50,000 rows	0.8748

60,000 rows	0.8784
70,000 rows	0.8811
80,000 rows	0.8834

Here is an example of running the augmentation application on a row from this dataset.
Test line:
__label__2 Amazing!: LJ smith is a fantastic author who inspires others to create stories and poetry that shows their true brilliance. Daughters of Darkness has a special quality about it which induces feeling of love and adventure all at the same time. I would not be surprised if people respond well to this novel!

Augmented Sentences:
__label__2 Amazing!: LJ smith is a bfantastic author who inspires others to crePate stories and potry tHhat shows their true brilliance. DDaughters of DarknUess has a speciall quality about itwhcih indduces feeling of lovee andud adventure all at the same time. I would not be surprised if people respond welml to this novvel!
__label__2 Ammazing!: LJ smith is a bfantastic author wOho inspires others to crePate torries aand potry tHHhat shows their rtue brilliance. DDaughterds of darknuess has a speciall uality about itwhhcih indduces feeling of lovee andud adventture all at the same time I would noQt be surprised if people respond welml to this novvel!
__label__2 Ammazing!: LJ smith is a bfan6astic author wOhk inspires others to crePate torrixes aaand ppotry ttHHhat shows their wtue brilliance. DDaughterds respond darknuess has a specall uality aboutt itwhhcih indduces feeling of lovde andud adventture all at the same time I would noAQt be surprised if people of welml to this novvvel!

Result

10,000 rows

version	Precision
1	0.7846
2	0.8290
3	0.8356
1+original	0.8398
2+original	0.8426
3+original	0.8436

20,000 rows

version	Precision
1	0.8420
2	0.8553
3	0.8585
1+original	0.8601
2+original	0.8604
3+original	0.8614

The best precision result was for 20,000 rows + original, version =2. So we take that row and test the performance when applying different frequency of perturbations for sentence and word parameters.

Word Perturbation Frequency	Sentence Perturbation Frequency	Precision
0.1	0.1	0.8613
0.1	0.2	0.8612
0.2	0.1	0.8625
0.2	0.2	0.8604

0.3	0.3	0.8616

The best precision is taken (20,000 rows + original, version =2, Word Perturbation Frequency=0.2, Sentence Perturbation Frequency=0.1) and this time different values of repeat_num is tested for word and sentence perturbation.

Word Perturbation Repeat	Sentence Perturbation Repeat	Precision
1	1	0.8625
1	2	0.8605
2	1	0.8612
2	2	0.8620

Language Detection

Overall, the data consisted of 6,00,000 rows [12]. I took ten percent for test dataset (600,000) and the rest was used for training data. It is important to note that for this dataset the typo_qwerty as well as add_letter functions were disabled, since they were depending on the language of the dataset to remain the same.

The python code to run is displayed below.

```
fasttext supervised -input train.txt -output langdetect -dim 16
```

The original precision for different row size of training data is show below.

	Precision
10,000 rows	0.7081
20,000 rows	0.7691
30,000 rows	0.8076
40,000 rows	0.8285
50,000 rows	0.8437
60,000 rows	0.8518
70,000 rows	0.8590
80,000 rows	0.8630

Here is an example of running the augmentation application on a row from this dataset.

Test line:

__label__eng Tom looked into his knapsack and realized that his computer had been stolen.

Augmented Sentences:
__label__eng Tom looked into his knapsack and realizeed that his compputer had been stolen.

__label__eng Tom looked into hSis knnavpsack and reelizead that h8s compputer had been stolen.

__label__eng om looked into hSis knnavpsack and reelizead that h8s compputer had beeen stlen.
Result

10,000 rows

version	Precision
1	0.6532
2	0.7401
3	0.7656
1+original	0.7535
2+original	0.7867
3+original	0.7997

20,000 rows

version	Precision
1	0.7481
2	0.8031
3	0.8174
1+original	0.8158
2+original	0.8338
3+original	0.8426

20,000 rows + original, version=3

Word Perturbation Frequency	Sentence Perturbation Frequency	Precision
0.1	0.1	0.8487
0.1	0.2	0.8491
0.2	0.1	0.8462
0.2	0.2	0.8426
0.3	0.3	0.8366

20,000 rows + original, version =3, Word Perturbation Frequency=0.1, Sentence Perturbation Frequency=0.2

Word Perturbation Repeat	Sentence Perturbation Repeat	Precision
1	1	0.8491
1	2	0.8477
2	1	0.8444
2	2	0.8425

StackExchange Cooking Question

Overall, the data consisted of 15,404 rows [13], which was a relatively small dataset. I took approximately ten percent for test dataset (1,500) and the rest was used for training data. It is important to note that for this dataset the precision and recall value are different and therefore the measure for performance is F-Score.

The python code to run is displayed below.

```
fasttext supervised -input cooking.train -output model_cooking -lr
1.0 -epoch 25
```

The original result for different row size of training data is show below.

	Precision	Recall	F-Score
5,000 rows	0.4623	0.2	0.2792
10,000 rows	0.5293	0.2284	0.3191
All (13,904 rows)	0.5660	0.2439	0.3409

Here is an example of running the augmentation application on a row from this dataset.

Test line:

__label__sauce __label__cheese How much does potato starch affect a cheese sauce recipe?

Augmented Sentences:

__label__sauce __label__cheese How much does po4aot starch affect a ceese sauce recipe?

__label__sauce __label__cheese Ho much dooes po4aot starch affecta ceesie zaucDe recipe

__label__sauce __label__cheese Ho much dooes po4aot starch affecta TceesiezaucDe recipe

Result

5,000 rows

version	precision	recall	F-score
1	0.4310	0.1864	0.2603
2	0.4363	0.1887	0.2635
3	0.4230	0.1830	0.2554
1+original	0.4590	0.1986	0.2772
2+original	0.4530	0.1960	0.2736
3+original	0.4523	0.1957	0.2732

10,000 rows

version	precision	recall	F-score
1	0.492	0.2123	0.2966
2	0.5047	0.2177	0.3042
3	0.4853	0.2094	0.2925
1+original	0.5107	0.2203	0.3078
2+original	0.516	0.2226	0.3110
3+original	0.506	0.2183	0.3050

10,000 rows + original, version =2

Word Perturbation Frequency	Sentence Perturbation Frequency	precision	recall	F-score
0.1	0.1	0.5193	0.2240	0.3130
0.1	0.2	0.5107	0.2203	0.3078
0.2	0.1	0.5133	0.2215	0.3094
0.2	0.2	0.516	0.2226	0.3110
0.3	0.3	0.5133	0.2215	0.3094

10,000 rows + original, version =2, Word Perturbation Frequency=0.1, Sentence Perturbation Frequency=0.1

Word Perturbation Repeat	Sentence Perturbation Repeat	precision	recall	F-Score
1	1	0.5193	0.2240	0.3130
1	2	0.504	0.2174	0.3038
2	1	0.516	0.2226	0.3110
2	2	0.5087	0.2194	0.3066

Chapter Five: Conclusion

Summary

This study provides a new tool for data scientists to modify their datasets by applying realistic augmentations and improve the performance of their models. The program includes several functions, mimicking the errors and noise, which can be in text, and includes variables that can be set up to control the aggressiveness of those functions. Comparison with existing models on text classification datasets showed that by providing different version of same rows, the precision would improve.

Future Work

For future work, I would suggest the following improvements of the thesis and project:

- Implement grid search to figure out the effect each perturbation has on the overall performance of the application
- Support more dataset file types
- Support more languages/ keyboard layouts
- Improve speed for application
- Test application with more datasets, e.g. unsupervised learning

References

[7] Bittlingmayer, Adam Mathias. “Amazon Reviews for Sentiment Analysis | Kaggle.” *Countries of the World | Kaggle*, 24 May 2017, www.kaggle.com/bittlingmayer/amazonreviews.

[9] “Downloads.” *Download Sentences - Tatoeba*, tatoeba.org/eng/downloads.

[13] Facebookresearch. “Facebookresearch/FastText.” *GitHub*, github.com/facebookresearch/fastText/blob/master/docs/supervised-tutorial.md#getting-and-preparing-the-data.

[2] “Image Preprocessing.” *Keras Documentation*, keras.io/preprocessing/image/.

[12] “Language Identification · FastText.” *FastText*, fasttext.cc/blog/2017/10/02/blog-post.html.

[5] “List of Options · FastText.” *FastText*, fasttext.cc/docs/en/options.html.

[1] “Machine Learning Crash Course | Google Developers.” *Google*, Google, developers.google.com/machine-learning/crash-course/.

[10] “Optimizing Your Code Using Profilers.” *JetBrains*, www.jetbrains.com/help/pycharm/profiler.html.

[11] “Packaging and Distributing Projects.” *Packaging and Distributing Projects - Python Packaging User Guide*, packaging.python.org/tutorials/distributing-packages/.

[8] “Seasoned Advice.” *Seasoned Advice*, cooking.stackexchange.com/.

[6] “Unicode Characters in the 'Punctuation, Other' Category.” *Hash: Online Hash Value Calculator*, www.fileformat.info/info/unicode/category/Po/list.htm.

[4] “FastText.” *Facebook Research*, research.fb.com/fasttext/.

[3] Billa, Jayadev. “Improving LSTM-CTC based ASR performance in domains with limited training data”, 2017