Azerbaijani-Unicodification

Team

Developer

Mark Gerken

Professor:

Kevin Scannell, PhD.

Requirements

- Python 3.3+ (Tested on Python 3.6.4)
- Unidecode 1.0.22
- unidecodecsv 0.14.1

Set up

1. Install Python
2. If not installed, install Virtualenv with `pip install virtualenv`
3. Navigate in a terminal window to the root directory of the project and set up a virtual environment with `virtualenv venv`
4. Activate the virtual environment with `venv\Scripts\activate` on a Windows machine or `soruce venv\bin\activate`
   - *More complete directions can be found here*
5. Install the required Python packages with `pip install -r requirements.txt`

Testing

Training data received from `ajz-train.txt`. Text file contains large corpus of Azjerbaijani text with (presumed) correct diacritics. Program trains off of this data, then receives `input.csv`. This CSV file contains lines of non-decorated Azjerbaijani text and attempts to restore all necessary diacritics. The program outputs the file `prediction.csv`. This file is then uploaded to Kaggle for scoring.

Process

Attempt #1: Dictionary Style

Memorized each word in training data. If found in test data, replaced word with example seen in training data.

Result: 0.60232

Attempt #2/3: Single letter Unigram

Records each instance of a unique character. Every character is replaced individually with the most recently seen equivalent, diacritically decorated or not. Attempt #3 was the fix of a logic error, which caused every single character to be replaced with a decorated character.

Result: ~~0.12493~~ 0.53447

Attempt #4: Dictionary and Frequency List

Combines previous attempts in filter-like style approach. Code has been structured to support multi-line contexual analysis (not yet implemented as of 15 April 2018). Conditionals test if line has been seen before in dictionary style, otherwise defaults to evaluating each letter against the frequency list described above.

Result: 0.84613

Attempt #5/6: 5/4/3 Letter Recognition, Dictionary, and Frequency List

Filter-like approach expanded upon. After comparing word to dictionary, each word is analyzed for a before seen 5/4/3 letter pattern. The word will undergo any replacements necessary, and then be passed through the frequency list comparison. Capitalization is still preserved. ~~Program reports never finding a letter pattern it hasn't seen in training, so suspecting bug.~~ Logic error resolved (or at least improved). Program is currently very inefficent. Processing requires several minutes (recorded at about 2:20) to complete. Will be looking into storing processed lists externally instead of fully recreating each time.

Result: ~~0.83943~~ 0.85174

Attempt #7: Two and Three Line Context

Completed dictionary to file processing, drastically reducing run time of program. All text processing is now saved to CSV files and read back. If CSV files are not present, program will simply redo the processing. Attempted incorporating multi-line comparisons for context. Processing time begam impossibly large, and could not devise way to reduce efficiently. All necessary code was migrated to a single file, documented appropriately, and removed from program functionality.

Result: No submission

Attempt #8: Minor bug fixes and short string resolution

Program would skip short strings, as well as cut out lines with commas. Logic and bug fixes. Program cleaned up in general. Context functionality removed.

> Result: 0.90558

Attempt #9/10: Handled weird ASCII characters, fixed default resolution

Program would return empty strings on ASCII characters not encountered before, now resolved. Changed order of restoration to Dictionary -> 5 Letter -> 4 Letter -> 3 Letter -> Frequency List -> Default. Needs functionality for allowing multiple multiLetter replacements in a single line. Attempt #10 involved restoring lines with quotes to see if possible error came from only commas being surrounded by quotes. Made no affect. Unsure why percentage went down by two percent.

> Result: ~~0.88577~~ 0.88577