Introducing FLIGHT

A **LIGHT**weight implementation of Dr. **F**ill Through Extreme Text Classification

Seamus Gould, Rowan Roshong

May 10, 2022


ERA
ONE
AREA
ALA
END
ORE
ALE
ERR
ERE
ATE
EAR
ANTE
ELI
EEL
ERIE
ARIA
SPA
ETA
ODE
SEA
ACE
TEE
ALI
ART
IRA
EST
ASS
SEE
NEST
IDEA
ESS
ETNA
ADE
URGE
ROE
RED
AGO
OGRE
IOTA
ABET
OLEO
LEI
ETAL
OSLO
ASEA
INN
ISEE
ARENA
IDOL
KNEE
ACHE
ERO
TOE
ARC
AHA
ILL
EERIE
ERIN
EMU
STET
ARI
TEN
NEAR
ASK
ODD
ASHE
ADAM
ELL
UNIT
RAT
OAT
EVEN
TEEN
NERO
ANON
RTE
EGG
AMI
ALTO
SIR
ACRE
EDEN
DEN
ACT
ELM
OMAR
ELLA
ALEE
SLED
EDGE
ASH
ELAN
ELK
ELIA
ONO
OREO
EAT
ANTI
IRAN
ERASE
LIE
ENE
ISLE
STEP
OPERA
EPIC
ANNE
EN
ADD
STEM
EPEE
OLD
OIL
AIL
ALEC
LEA
OMIT
OPEN
EMIR
EGO
EDIT
REST
OPAL
TSAR
LET
EVIL
EBB
YE
AMEN
LEE
USE
IRON
EWE
ESE
ESTE
ETON
ETE
OAR
AIR
ATOM
OTTO
APE
ECHO
EAST
NEED
TNT
STIR
TIE
EMIT
USER
TREE
ARAB
ALAN
ARM
TAR
TEST
ASIA
AURA
ENO
ABE
ICE
IDO
ANA
EYE
IRE
OMEN
OBOE
AGE
ESSE
IDLE
OLE
ALP
ION
ASP
SET
UTE
APSE
PEA
NEE
AMO
AMA
SST
AVE
ITEM
ANN
ORAL
ADO
NET
EON
ENTER
TEA
EASE
RIO
STY
ODOR
EVE
URN
STAR

Table of contents

Introduction

Background

One common benchmark for artificial intelligence comes from its ability to play games.

1. In 2007 researchers weakly solved checkers [1]
2. Deep Blue defeated Kasparov in a six game match in 1997. [2]
3. The first example of a crossword defeating top ranked crossword solvers only happened last year By Dr. Fill.

[1] https://en.wikipedia.org/wiki/Checkers
[2] https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov

Dr. Fill

- Created by Matt Ginsberg and developed since 2012 that competed against human solvers.
- The program defeated all human participants in 2021 completing the final puzzle in just 49 seconds.
- Matt Ginsberg thought about the program as a Constraint Satisfaction Problem, in short, maximize the probability of a puzzle fill being correct given the clues and board.

Extreme Text Classification

This is not just any classification problem. Naive Bayes, general neural networks, and other classifiers cannot deal with memory requirements to classify such a dataset.

Text classification of millions of classes is a branch of classification has only existed in the past decade. It is most commonly done with hierarchical softmax loss function.

Dr. Fill used dense passage retrieval to retrieve documents from the web to create a set of candidate words, and then used a limited discrepancy search to maximize the likelihood that it completes the puzzle. We used decided on approaching the problem as a text classification problem. We did this for a few reasons.

- The number of out of vocabulary words are minimal.
- The speed for inference is much faster.
- We wanted a lightweight version that was memory and computationally inexpensive. We thought this would be a better approach than dense passage retrieval.

Our Goal

Our Goal

Our goal was to make a version of Dr. Fill that was lightweight and fast. Eventually we found that we could make one that is not just lightweight, but also competitive with Dr. Fill.

What We've Done So Far

Early Approach

In the beginning, we attempted to use dense passage retrieval to get the probability of candidate words.

- Concatenate each clue into a ”document” for word
- Remove stop words and lemmatize each clue then hash into an array.
- Treat a clue like a query, hash it into an array and take the dot product of that array and the matrix of TFIDFs.
- Rank the highest words according to the computation.
- Computationally slow.
- Precision only achieved fifteen percent.

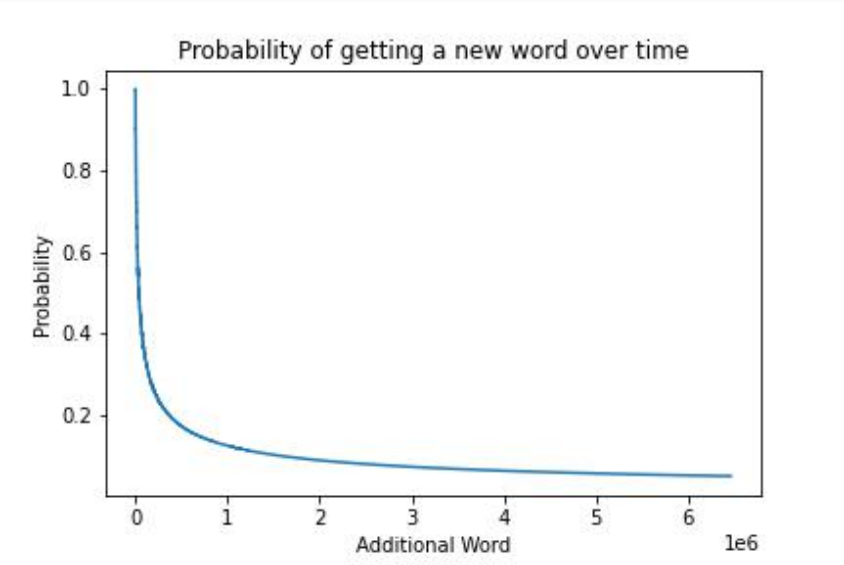
Extreme Text Classification

After finding Manik Varma's repository [3] of extreme text classification, we experimented using several different algorithms. We tried using all possible models that were given (there were dozens!). The only two models that actually worked for us were FastText and probabilistic label trees from the extremeText library. We preferred using FastText because we were able to train it with a test set unlike a extremeText, which did not allow us to do this. Also, FastText had better documentation.

[3] http://manikvarma.org/downloads/XC/XMLRepository.html

Probability of getting a new word

One fear that we had was that a word could not exist in our training data. Our classification does not take into consideration out of vocabulary clues. One can see that this is unlikely. The likelihood of getting a new word is equivalent to $\frac{\text{number of distinct words}}{\text{number of total words}} = .048$



Building The Model

We ran a fasttext model by dividing the train, test, and validation set into 98, 1, 1 percent respectively. We fed the test set into fasttext as well and evaluated the performance on the validation set. In addition, we made the loss function equal to hierarchical softmax, which enables us to classify thousands of classes.

We decided to build our Fasttext model with a dataset compiled by Saul Pwanson. It contained six and a half million clues, and there where three hundred and fifteen thousand distinct words.

Greedy Approach

Greedy Approach

Our first stab at making a crossword solver was to use a greedy approach. This mostly works in three steps:

1. Apply a threshold of .99 to our model, fill in every across and down clue above threshold, drop threshold by .01 and repeat.
2. Apply a threshold of -1 to our model and fill in the rest of what we can
    a. A threshold of -1 returns all possible answers our model believes could fit the clue.
3. Fill in the few remaining blanks.

Branching/Pruning Approach

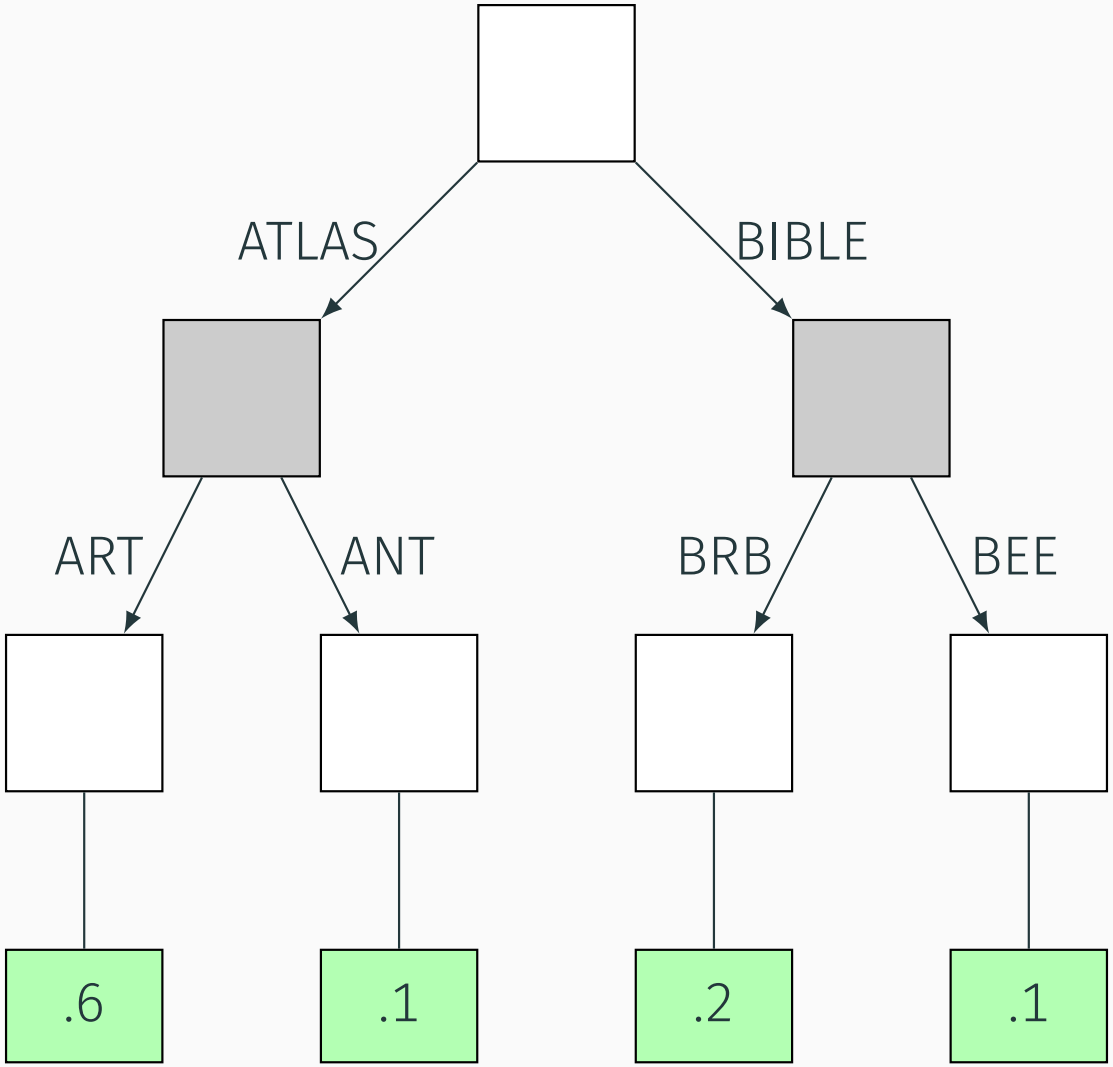
Branching/Pruning Approach

Crossword Solving with Branching and Pruning

procedure SolveCrossword(clues, [(board, 1)])

```
1:  procedure addWord([(pct, boards)])          ▷ likelihood and board
2:      leafs = []
3:      if board not complete then
4:          for Each board in list of boards do
5:              Add one word
6:              Append output to leafs
7:          end for
8:          if The list is null then
9:              leafs = boards                              ▷ boards
10:         end if
11:         pct *= prob
12:         leafs = PruneBranches(leafs)
13:         return SolveCrossword(leafs)    ▷ Applied recursively
14:
```

Branching/Pruning Approach



What We Need to do Next

Fill In The Blanks

Our model struggles with recall. When running our solvers on puzzles that come later in the week, they often struggle to fill in out-of-vocabulary words.

This is not typically an issue because every letter fits into two clues, so if it can't figure out the across, just try the down. This does not always work, so we need a way to fill in the few remaining blanks that exist at the end of solving.

Fix Wrong Answers (Better)

We also have the issue of our model picking the wrong word. This is where our greedy approach takes a wrong turn, so this is why we delved into branching, which has worked okay so far. We need to improve this system.

Questions?

Thank You!

Thank you for listening.