TREXQUANT - HANGMAN CHALLENGE SOLUTION

~ ANSHUMAN MONDAL (IITG)

A dynamic, heuristic-driven algorithm to solve the hangman problem is described below:

1. First I try to evaluate the frequency of vowels in a word (how often they repeat) and remove all newline characters in the words_250000_train.txt given as a part of the task. This acts like a feature extraction step. A histogram is plotted to visualize the empirical spread of vowel ratios, to be able to choose a threshold.
2. ***N-gram word segmentation:*** Then, construct a dictionary (n_word_dictionary) that looks for patterned substrings to match / assist in solving the hangman problem.

Working Strategy:

1. Primary Match- Regex filtering converts input word (_pp_e) to (.pp.e) format. Filters current candidate using re.match()
2. Secondary Step: Picks the most frequent unguessed letter, other than vowel if its density is low. Incase it fails it uses func2() to find matching pairs in dictionary. Applies sorting based on frequency. Threshold of 0.55 is chosen after studying the histogram plot to be safe for vowel heuristic.
3. Final touch: Multi level substring matching is done by choosing differential lengths / chunks of substrings equal to 0.75, 0.50. 0.33 times that of the word length. This takes care of various length of words, substrings smaller than 3 in size are insignificant and would contribute to noise and inaccuracies in the guessing process so we could neglect them safely, unless we have to predict really small words that could be a possible loophole of the approach.
4. Incase, all the above steps fail guessing is done using global letter frequency from the full dictionary to go with the most occurring unguessed letter so far.

Why this approach could be a **great starting point** to other ML architectures?

1. Regex and n-gram matching simulate pattern learning which is automated by neural models.
2. These are hand-crafted versions of what ML models learn directly from data.