COURSE: COMP 4475

PROJECT 2 – EXPERIMENTATION WITH WORD VECTORS IN JULIA

NAME: DHAVAL THANKI

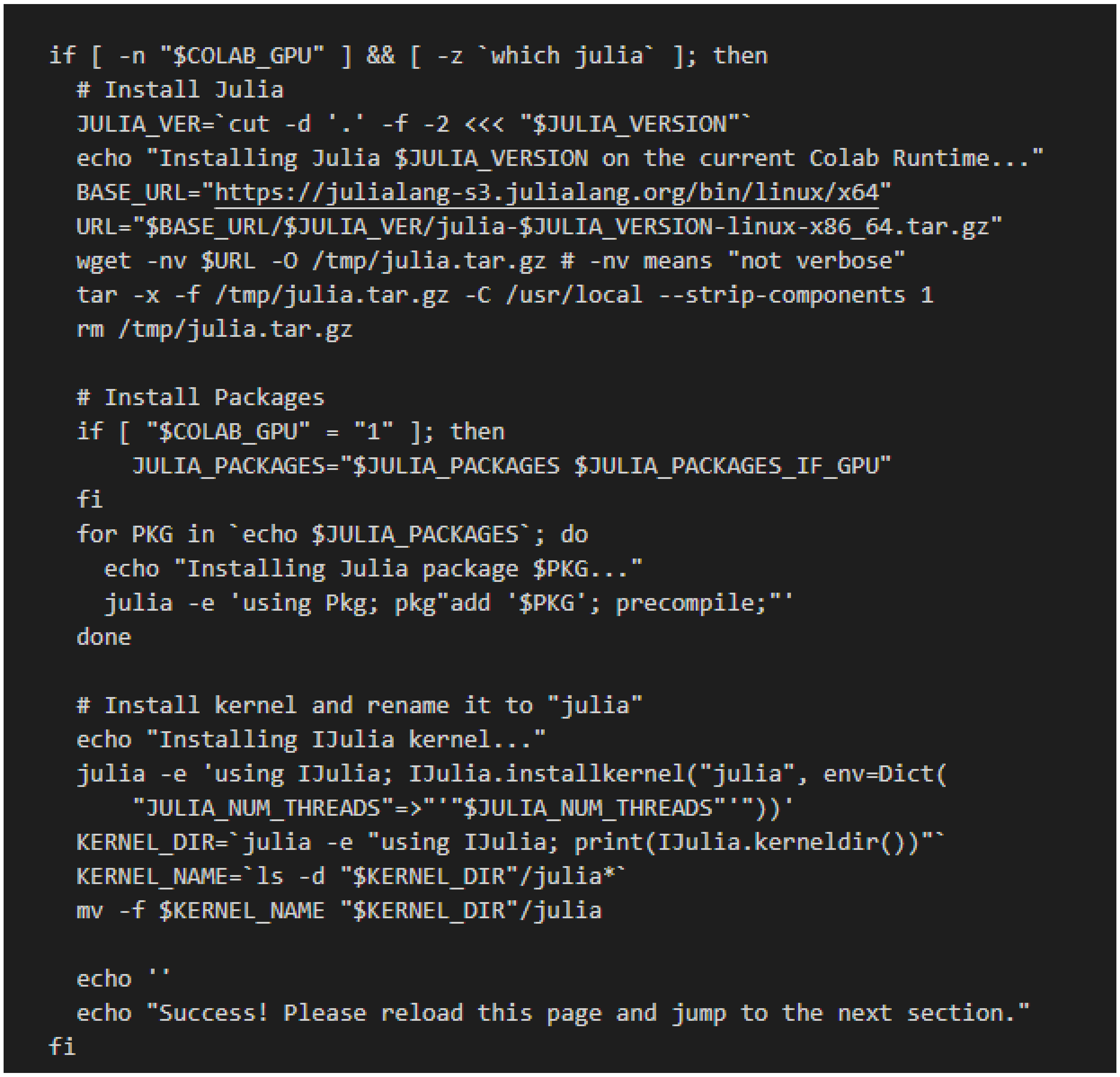
STUDENT NUMBER: 0871347

Objective

In this project, we are asked to dive into getting acquainted with word vectors and word embedding utilities in Julia, it is interesting to see the relations formed between words using vectors and mathematical computations to find similarities and dissimilarities between like and unlike words. Here are the source files for easy access, Comp 4475 - Project 2 - 0871347.ipynb, Comp 4475 - Project 2 - 0871347.jl

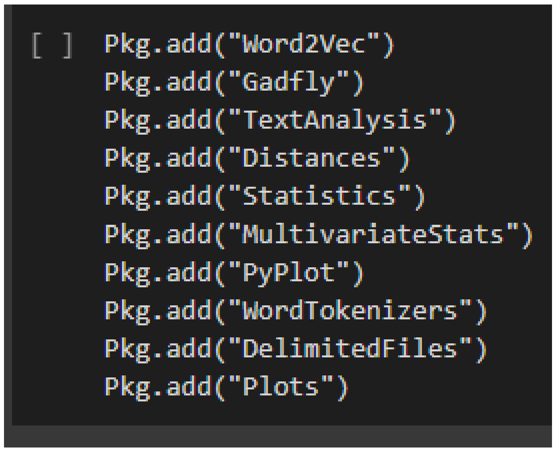
Running Julia on Colabs

Since I couldn't get Word2Vec to run on my windows machine, and I didn't want to create a virtual machine running linux for this sole purpose, I turned to colabs so that I can utilise this API. Here is the screenshot of how to get it to run on there

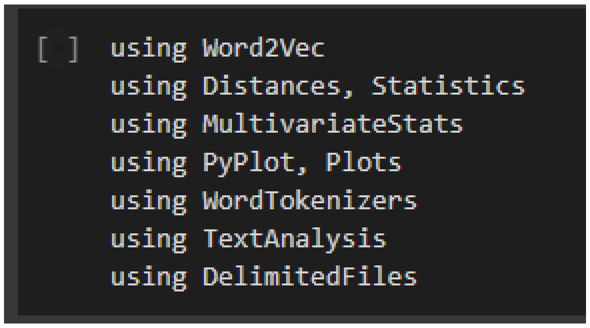


```
if [ -n "$COLAB_GPU" ] && [ -z `which julia` ]; then
  # Install Julia
  JULIA_VER=`cut -d '.' -f -2 <<< "$JULIA_VERSION"`
  echo "Installing Julia $JULIA_VERSION on the current Colab Runtime..."
  BASE_URL="https://julialang-s3.julialang.org/bin/linux/x64"
  URL="$BASE_URL/$JULIA_VER/julia-$JULIA_VERSION-linux-x86_64.tar.gz"
  wget -nv $URL -O /tmp/julia.tar.gz # -nv means "not verbose"
  tar -x -f /tmp/julia.tar.gz -C /usr/local --strip-components 1
  rm /tmp/julia.tar.gz

  # Install Packages
  if [ "$COLAB_GPU" = "1" ]; then
      JULIA_PACKAGES="$JULIA_PACKAGES $JULIA_PACKAGES_IF_GPU"
  fi
  for PKG in `echo $JULIA_PACKAGES`; do
    echo "Installing Julia package $PKG..."
    julia -e 'using Pkg; pkg"add '$PKG'; precompile;"'
  done

  # Install kernel and rename it to "julia"
  echo "Installing IJulia kernel..."
  julia -e 'using IJulia; IJulia.installkernel("julia", env=Dict(
      "JULIA_NUM_THREADS"=>"'"$JULIA_NUM_THREADS"'"))'
  KERNEL_DIR=`julia -e "using IJulia; print(IJulia.kerneldir())"`
  KERNEL_NAME=`ls -d "$KERNEL_DIR"/julia*`
  mv -f $KERNEL_NAME "$KERNEL_DIR"/julia

  echo ''
  echo "Success! Please reload this page and jump to the next section."
fi
```

Adding and stating what packages are going to be used



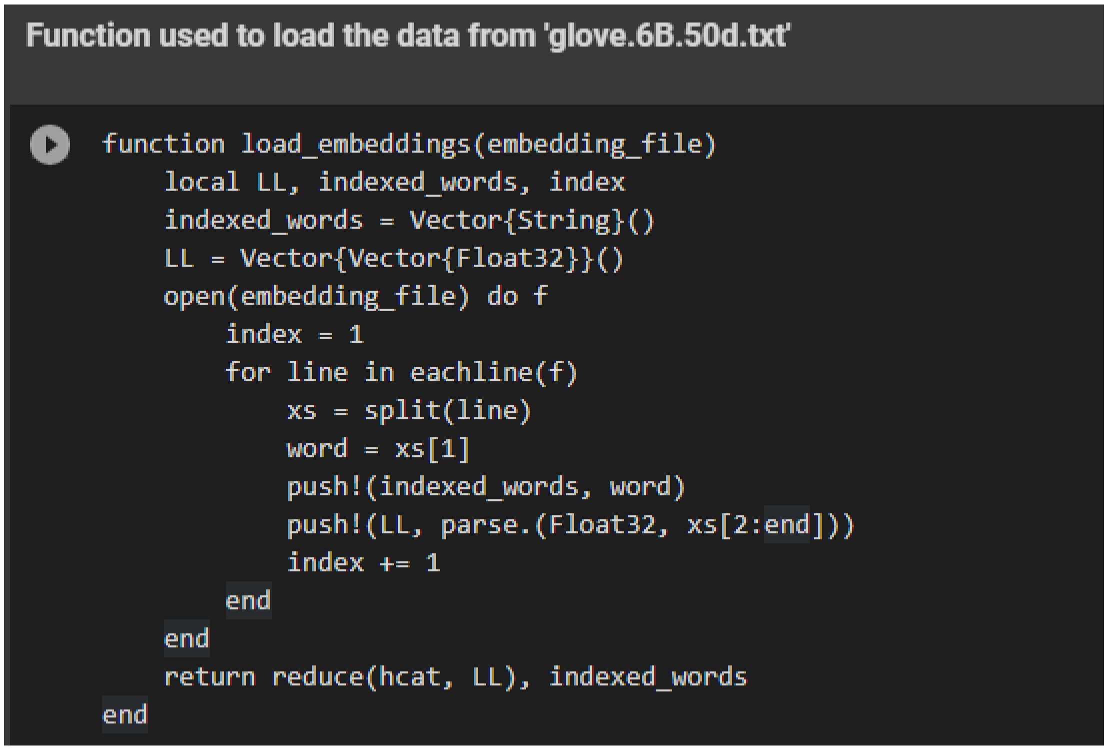
```
[ ]  Pkg.add("Word2Vec")
     Pkg.add("Gadfly")
     Pkg.add("TextAnalysis")
     Pkg.add("Distances")
     Pkg.add("Statistics")
     Pkg.add("MultivariateStats")
     Pkg.add("PyPlot")
     Pkg.add("WordTokenizers")
     Pkg.add("DelimitedFiles")
     Pkg.add("Plots")
```



```
[ ]  using Word2Vec
     using Distances, Statistics
     using MultivariateStats
     using PyPlot, Plots
     using WordTokenizers
     using TextAnalysis
     using DelimitedFiles
```

Function defs

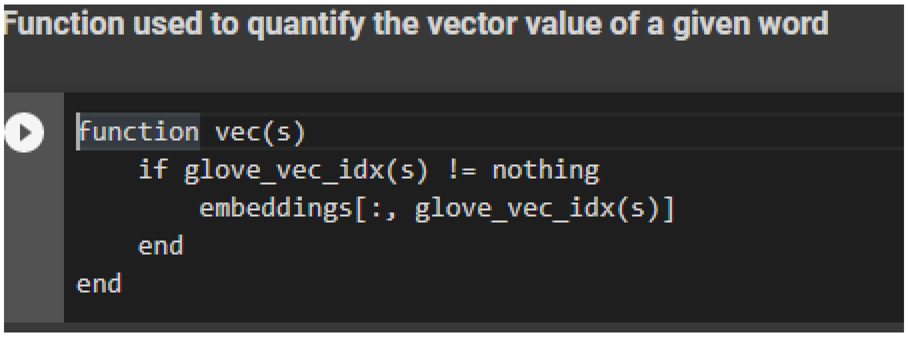
For the one-Hot-N encoder



Function used to load the data from 'glove.6B.50d.txt'

```
function load_embeddings(embedding_file)
    local LL, indexed_words, index
    indexed_words = Vector{String}()
    LL = Vector{Vector{Float32}}()
    open(embedding_file) do f
        index = 1
        for line in eachline(f)
            xs = split(line)
            word = xs[1]
            push!(indexed_words, word)
            push!(LL, parse.(Float32, xs[2:end]))
            index += 1
        end
    end
    return reduce(hcat, LL), indexed_words
end
```

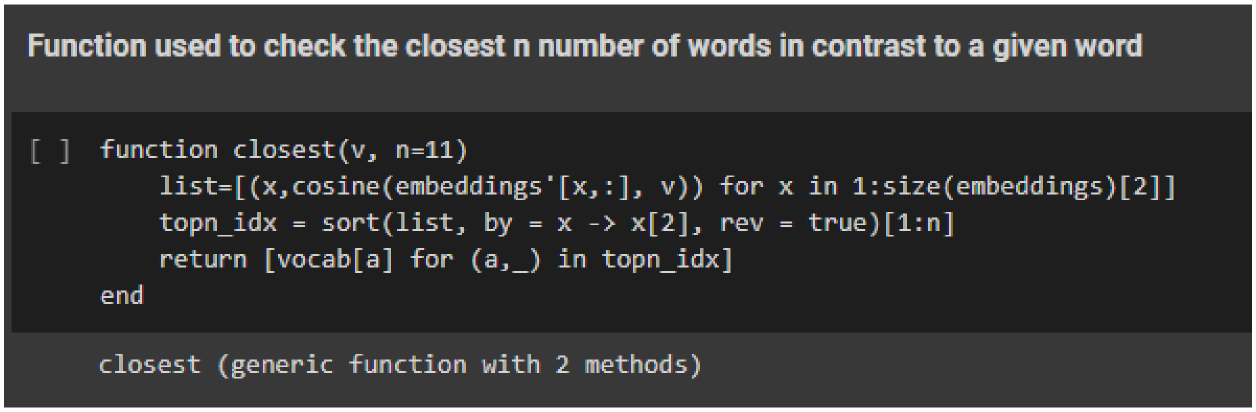
For the use of finding the index value of a word in vector form



Function used to quantify the vector value of a given word

```
function vec(s)
    if glove_vec_idx(s) != nothing
        embeddings[:, glove_vec_idx(s)]
    end
end
```

For finding the closest possible match to a given word

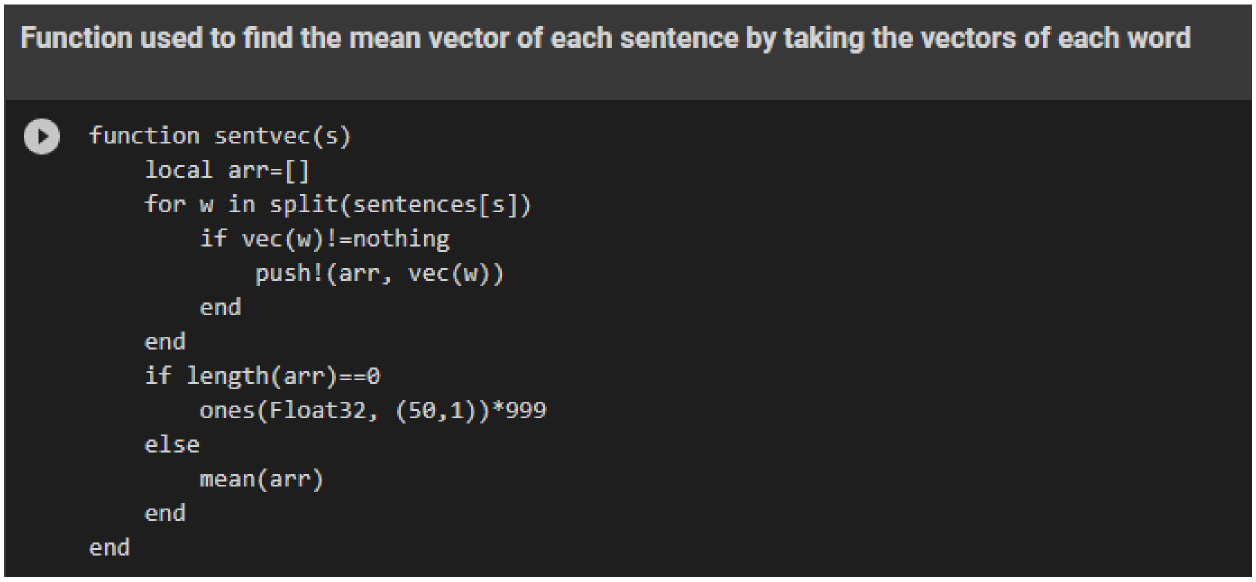


Function used to check the closest n number of words in contrast to a given word

```
[ ] function closest(v, n=11)
        list=[(x,cosine(embeddings'[x,:], v)) for x in 1:size(embeddings)[2]]
        topn_idx = sort(list, by = x -> x[2], rev = true)[1:n]
        return [vocab[a] for (a,_) in topn_idx]
    end
```

```
closest (generic function with 2 methods)
```

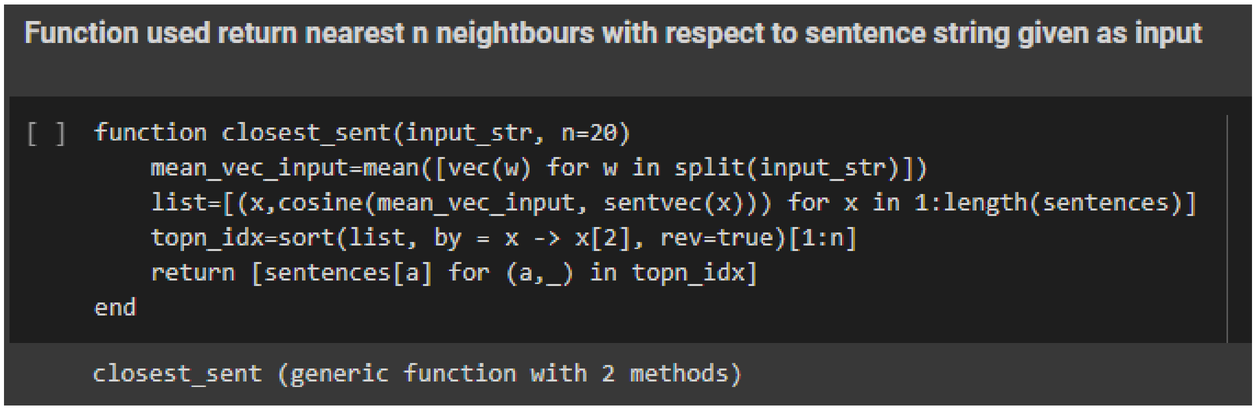
For evaluating the vector value of sentences by averaging each word-vector value, i.e the mean value in vector form of a sentence



Function used to find the mean vector of each sentence by taking the vectors of each word

```
function sentvec(s)
    local arr=[]
    for w in split(sentences[s])
        if vec(w)!=nothing
            push!(arr, vec(w))
        end
    end
    if length(arr)==0
        ones(Float32, (50,1))*999
    else
        mean(arr)
    end
end
```

For finding the closest sentence or string of words that match a given string based on the mean vector value of the input

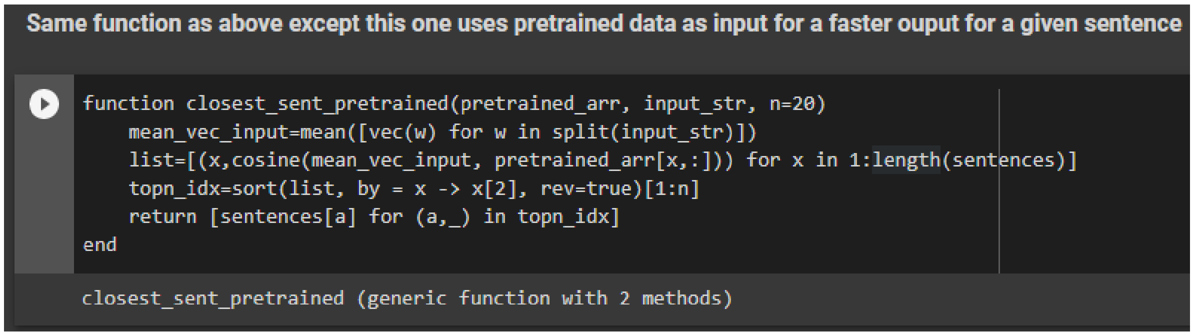


Function used return nearest n neightbours with respect to sentence string given as input

```
function closest_sent(input_str, n=20)
    mean_vec_input=mean([vec(w) for w in split(input_str)])
    list=[(x,cosine(mean_vec_input, sentvec(x))) for x in 1:length(sentences)]
    topn_idx=sort(list, by = x -> x[2], rev=true)[1:n]
    return [sentences[a] for (a,_) in topn_idx]
end
```

```
closest_sent (generic function with 2 methods)
```

Lastly, the same as above but in this case we use a pretrained dataset so we can speed up compilation time quite a bit, essentially a trained model.



Same function as above except this one uses pretrained data as input for a faster ouput for a given sentence

```
function closest_sent_pretrained(pretrained_arr, input_str, n=20)
    mean_vec_input=mean([vec(w) for w in split(input_str)])
    list=[(x,cosine(mean_vec_input, pretrained_arr[x,:])) for x in 1:length(sentences)]
    topn_idx=sort(list, by = x -> x[2], rev=true)[1:n]
    return [sentences[a] for (a,_) in topn_idx]
end
```

```
closest_sent_pretrained (generic function with 2 methods)
```

Creating the Models

We create numerous models from the text8 data file, these models comprise of a word-vector file, a wordphrase-vector file and a word cluster file. Each of these models are then described and used to describe the data they hold by checking similarity and dissimilarity with other words and using cosine evaluation to check what some basic mathematical equations will do to affect the result of their data.

It didn't seem right to put in 100 screenshots, so I made a PDF of the Jupyter notebook instead, it can be found here.

Once the models are created, a set of files are added to the Colab temp files section, these include a .txt for each of the aforementioned models, these files are then used to infer similarities between words.

I also used another data file named "glove.6B50d.txt", this is from the resource mentioned below, that resource helped me extensively in getting accustomed to working with sentences and comparing and analysing sentences rather than words or word phrases.

Optimised sentence evaluation

For optimising the sentence evaluation, we use a pretrained model based off the initial run of comparing the nearest similar sentences, the output of this is stored In a CSV file so that it may be used as a base for evaluating new input since we don't need to comb through the data again since the base data hasn't changed.

Resources used

https://spcman.github.io/getting-to-know-julia/nlp/word-embeddings/

https://github.com/JuliaText/Word2Vec.jl