End to End Question-Answering System Using NLP and SQuAD Dataset

LAVANYA S — Published On November 29, 2021 and Last Modified On August 24th, 2022

Advanced Guide NLP Python

Overview

My goal is to learn different NLP principles, implement them, and explore more solutions, rather than to achieve perfect accuracy. I've always believed in starting with simple models to gauge the level, and I've taken the same strategy here. This section will introduce Facebook sentence embeddings and how they may develop quality assurance systems.

Introduction Question-Answering System

Question answering is a critical NLP problem and a long-standing artificial intelligence milestone. QA systems allow a user to express a question in natural language and get an immediate and brief response. QA systems are now found in search engines and phone conversational interfaces, and they're fairly good at answering simple snippets of information. On more hard questions, however, these normally only go as far as returning a list of snippets that we, the users, must then browse through to find the answer to our question.

Reading comprehension is the ability to read a piece of text and then answer questions about it. Reading comprehension is difficult for machines because it requires both natural language understanding and knowledge of the world.

SQuAD Dataset for building Question-Answering System

The Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset made up of questions posed by crowd workers on a collection of Wikipedia articles, with the response to each question being a text segment, or span, from the relevant reading passage, or the question being unanswerable.

The reading sections in SQuAD are taken from high-quality Wikipedia pages, and they cover a wide range of topics from music celebrities to abstract notions. A paragraph from an article is called a passage, and it can be any length. Reading comprehension questions are included with each passage in SQuAD. These questions are based on the passage's content and can be answered by reading it again. Finally, we have one or more answers to each question.

One of SQuAD's distinguishing features is that the answers to all the questions are text portions, or spans, in the chapter. These can be a single word or a group of words, and they are not limited to entities–any range is fair game.

Question-Answering System

This is a very adaptable design, and we've found that it can ask for a wide range of queries. Instead of having a list of options for each question, systems must choose the best answer from all potential spans in the passage, which means they must deal with a vast number of possibilities. Spans have the extra benefit of being simple to evaluate.

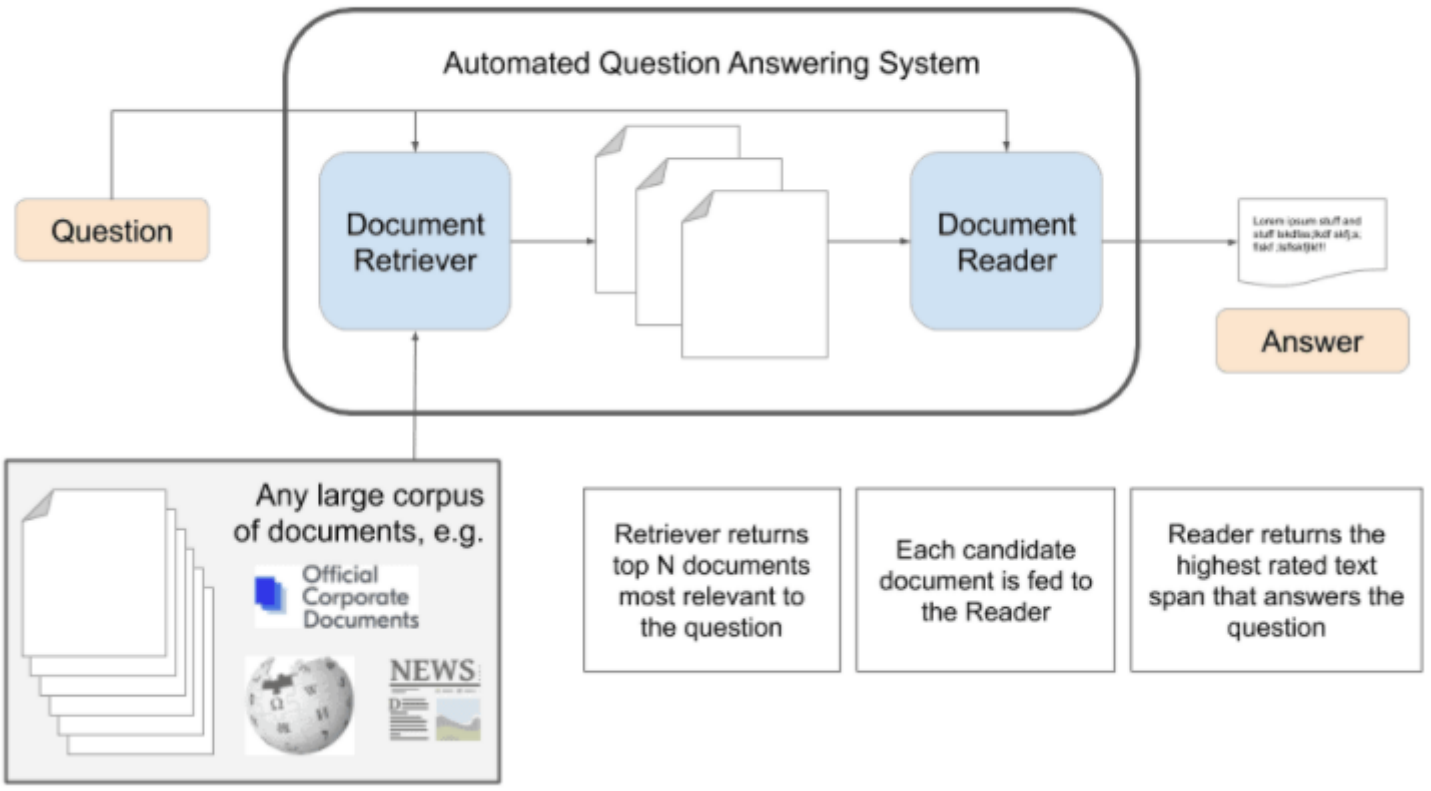
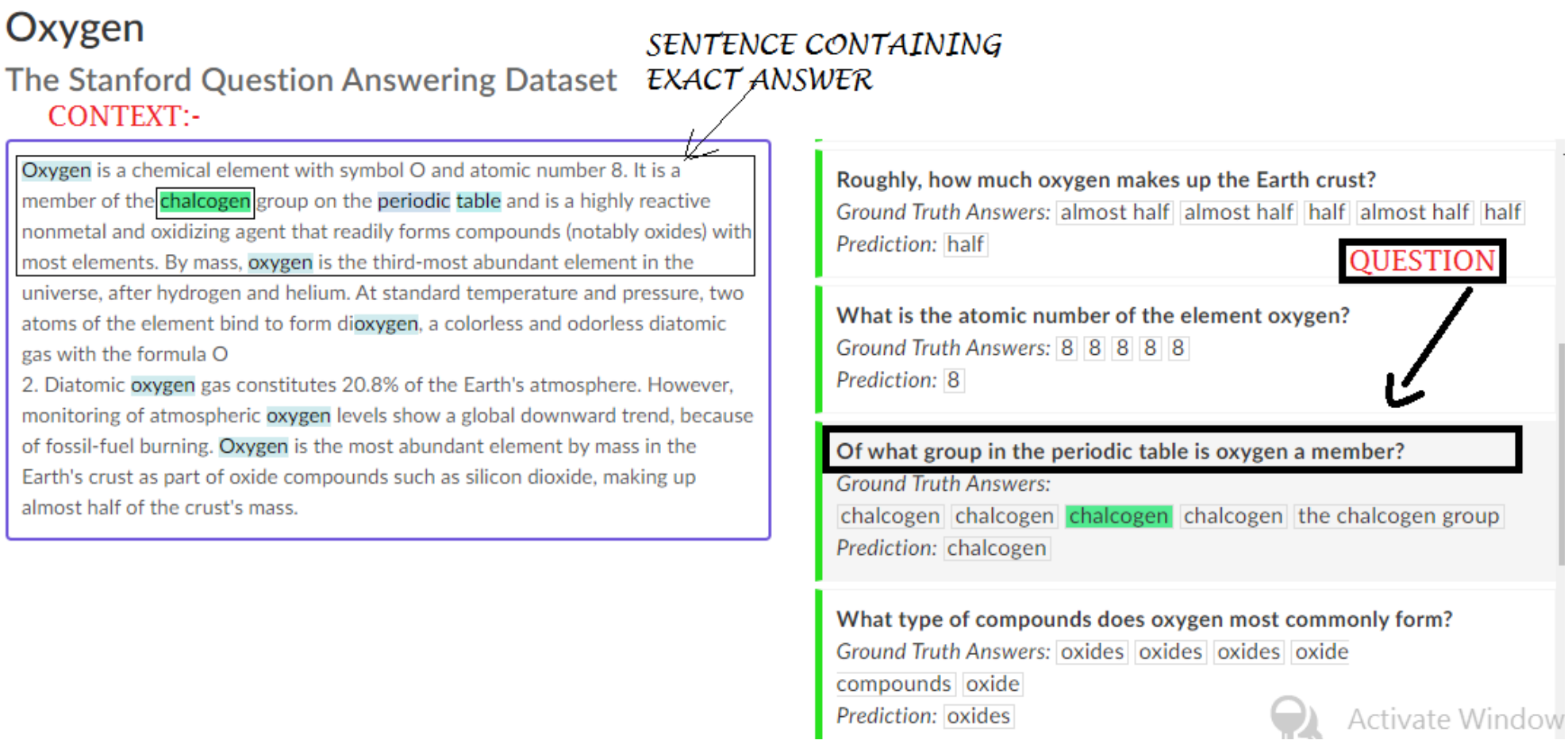


Image: Source

The QA setting, depending on the span is extremely natural. Open-domain QA systems can typically discover the right papers that hold the solution to many user questions sent into search engines. The task is to discover the shortest fragment of text in the passage or document that answers the query, which is the ultimate phase of "answer extraction."

Problem Description for Question-Answering System

The purpose is to locate the text for any new question that has been addressed, as well as the context. This is a closed dataset, so the answer to a query is always a part of the context and that the context spans a continuous span. For the time being, I've divided the problem into two pieces –



Source: SQuAT

Getting the correct solution to the sentence (highlighted green)

Getting the correct response from the sentence once we completed it (highlighted blue)

We have a context, question, and text for each observation in the training set. One such observation is:

Facebook Sentence Embedding

We now have word2vec, doc2vec, food2vec, node2vec, and sentence2vec, so why not sentence2vec? The main idea behind these embeddings is to numerically represent entities using vectors of various dimensions, making it easier for computers to grasp them for various NLP tasks.

Traditionally, we applied the bag of words approach, which averaged the vectors of all the words in a sentence. Each sentence is tokenized into words, and the vectors for these words are discovered using glove embeddings. The average of all these vectors is then calculated. This method has been done admirably, although it is not an accurate method because it ignores word order.

This is where Infersent comes in. It's a sentence embeddings method that generates semantic sentence representations. It's based on natural language inference data and can handle a wide range of tasks.

InferSent is a method for generating semantic sentence representations using sentence embeddings. It's based on natural language inference data and can handle a wide range of tasks.

The procedure for building the model :

Make a vocabulary out of the training data and use it to train the inferent model.

I used Python 2.7 (with recent versions of NumPy/SciPy) with Pytorch (recent version) and NLTK >= 3

if you want to download the trained model on *AllNLI* then run-

```
curl -Lo encoder/infersent.allnli.pickle
https://s3.amazonaws.com/senteval/infersent/infersent.allnli.pickle
```

Load the pre-trained model

```
import nltk
nltk.download('punkt')
import torch
        infersent = torch.load('InferSent/encoder/infersent.allnli.pickle', map_location=lambda
storage, loc: storage)
        infersent.set_glove_path("InferSent/dataset/GloVe/glove.840B.300d.txt")
        infersent.build_vocab(sentences, tokenize=True)
        dict_embeddings = {}
        for i in range(len(sentences)):
         print(i)
         dict_embeddings[sentences[i]] = infersent.encode([sentences[i]], tokenize=True)
```

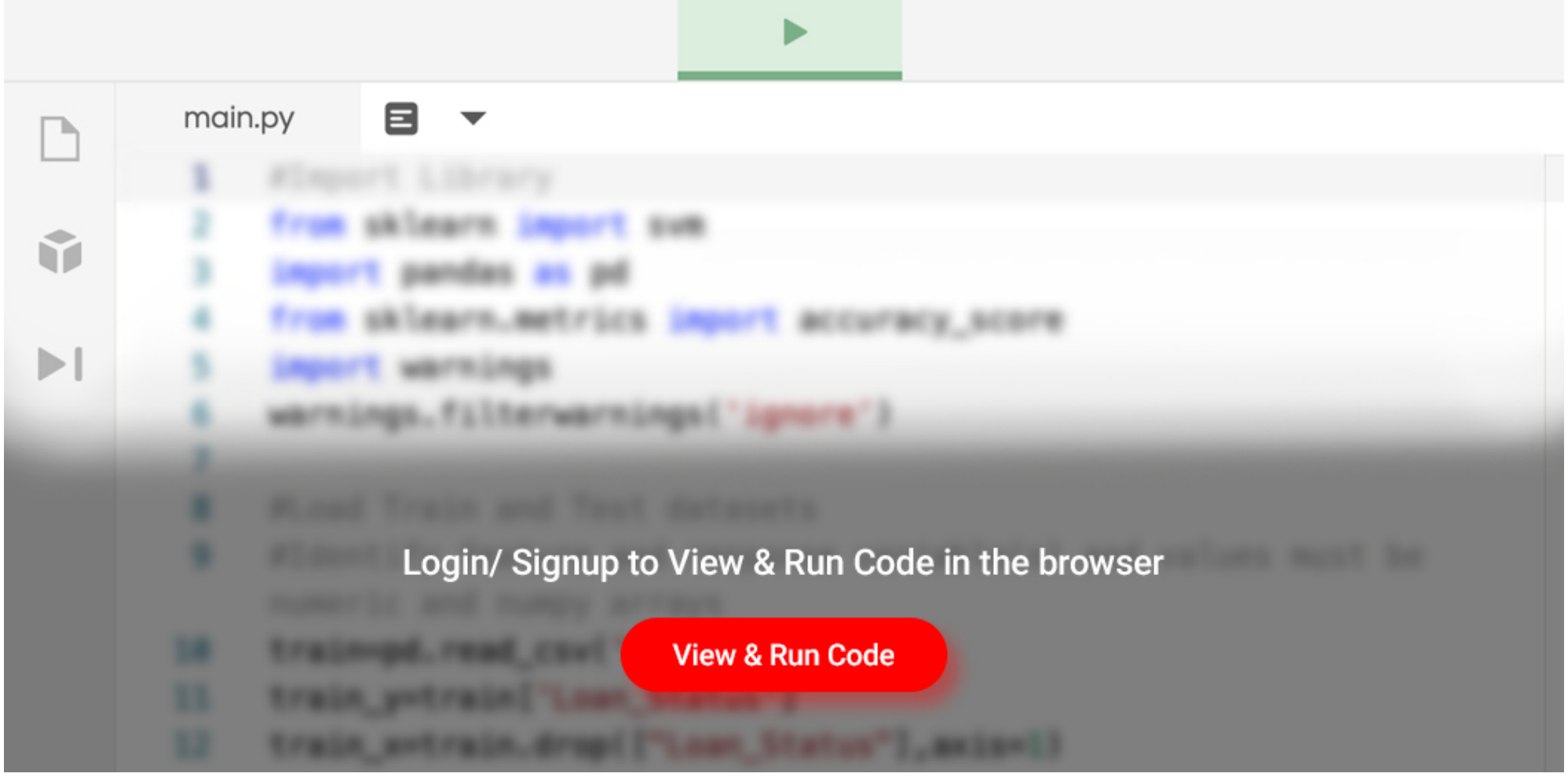
Where sentences are the number of sentences in your list. You can use infersent.update_vocab(sentences) to update your vocabulary or infersent. Build vocab k words(K=100000) to load the K most common English words directly. If tokenize is set to True (the default), NTLK will tokenize sentences.

We can use these embeddings for a variety of tasks in the future, such as determining whether two sentences are similar.

Sentence Segmentation:

You can use Doc.has_annotation with the attribute name "SENT_START" to see if a Doc has sentence boundaries. Here the paragraph is broken into a meaningful sentence.

Python Code:



Make many sentences out of the paragraph/context. Spacy and Textblob are two tools I'm familiar with for handling text data. TextBlob was used to do this. Unlike spacy's sentence detection, which can produce random sentences based on the period, it executes intelligent splitting. Here's a real-life example:

```
predicted["context"][21493]
```

```
"10th Street (40°44'03"N 74°00'11"W\ufeff / \ufeff40.7342580°N 74.0029670°W\ufeff / 40.7342580; -74.0029670) begins at
the FDR Drive and Avenue C. West of Sixth Avenue, it turns southward about 40 degrees to join the Greenwich Village s
treet grid and continue to West Street on the Hudson River. Because West 4th Street turns northward at Sixth Avenue,
it intersects 10th, 11th and 12th and 13th Streets in the West Village. The M8 bus operates on 10th Street in both di
rections between Avenue D and Avenue A, and eastbound between West Street and Sixth Avenue. 10th Street has an eastbo
und bike lane from West Street to the East River. In 2009, the two-way section of 10th Street between Avenue A and th
e East River had bicycle markings and sharrows installed, but it still has no dedicated bike lane. West 10th Street w
as previously named Amos Street for Richard Amos. The end of West 10th Street toward the Hudson River was once the ho
me of Newgate Prison, New York City's first prison and the United States' second."
```

The paragraph is splitting it into 7 sentences using TextBlob

```
['10th Street (40°44'03"N 74°00'11"W\ufeff / \ufeff40.7342580°N 74.0029670°W\ufeff / 40.7342580; -74.0029670) begins a
t the FDR Drive and Avenue C. West of Sixth Avenue, it turns southward about 40 degrees to join the Greenwich Village
street grid and continue to West Street on the Hudson River.',
 'Because West 4th Street turns northward at Sixth Avenue, it intersects 10th, 11th and 12th and 13th Streets in the
West Village.',
 'The M8 bus operates on 10th Street in both directions between Avenue D and Avenue A, and eastbound between West Str
eet and Sixth Avenue.',
 '10th Street has an eastbound bike lane from West Street to the East River.',
 'In 2009, the two-way section of 10th Street between Avenue A and the East River had bicycle markings and sharrows i
nstalled, but it still has no dedicated bike lane.',
 'West 10th Street was previously named Amos Street for Richard Amos.',
 "The end of West 10th Street toward the Hudson River was once the home of Newgate Prison, New York City's first pris
on and the United States' second."]
```

Here the sentence is split up into separate text :

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Environmentalists are concerned about the loss of biodiversity that will result from the
destruction of the forest, and also about the release of the carbon contained within the vegetation,
which could accelerate global warming.")
for token in doc:
    print(token.text)
```

```
Environmentalists
are
concerned
about
loss
of
biodiversity
that
will
result
from
destruction
of
the
forest
,
and
also
about
the
release
of
the
carbon
contained
within
the
vegetation
,
which
could
accelerate
```

Using the Infersent model, get the vector representation of each sentence and question.

Machine learning Models

we should tackle the problem by utilizing two key methods:

Supervised learning and unsupervised learning, in which I did not use the target variable. I'm going to return the sentence from the paragraph that is the furthest away from the given question.

Unsupervised Learning Model

Let's see if we can use Euclidean distance to find the sentence that is closest to the question. This model's accuracy was roughly 45 per cent. The accuracy rose from 45 per cent to 63 per cent after altering the cosine similarity. This makes sense because the Euclidean distance is unaffected by the alignment or angle of the vectors, whereas cosine is. With vectorial representations, the direction is crucial.

However, this strategy does not take advantage of the rich data with target labels we are given. However, because of the solution's simplicity, it still produces a solid outcome with no training. Facebook sentence embedding deserves credit for the excellent results.

Supervised Learning Model

Creating a training set for this section has been difficult since each portion does not have a predetermined amount of sentences and answers can range from one word to many words.

I've converted the target variable's text to the sentence index that contains that text. I've kept my paragraphs to a maximum of ten sentences to keep things simple (around 98 percent of the paragraphs have 10 or fewer sentences). As a result, in this scenario, I have 10 labels to forecast. I created a feature based on cosine distance for each sentence. If a paragraph has fewer than 10 sentences, I replace its feature value with 1 (maximum cosine distance) to make 10 sentences.

> Question – **What kind of sending technology is being used to protect tribal lands in the Amazon?**

> Context – ***The use of remote sensing for the conservation of the Amazon is also being used by the indigenous tribes of the basin to protect their tribal lands from commercial interests.*** Using handheld GPS devices and programs like Google Earth, members of the Trio Tribe, who live in the rainforests of southern Suriname, map out their ancestral lands to help strengthen their territorial claims. Currently, most tribes in the Amazon do not have clearly defined boundaries, making it easier for commercial ventures to target their territories.
> -From SQuAD

> **Text –** remote sensing

Because the highlighted sentence index is 1, the target variable will be changed to 1. There will be ten features, each of which corresponds to one sentence in the paragraph. Because these sentences do not appear in the paragraph, the missing values for column cos 2, and column cos 3 are filled with NaN.

The Amazon rainforest (Portuguese: Floresta Amazônica or Amazônia; Spanish: Selva Amazónica, Amazonía or usually Amazonia; French: Forêt amazonienne; Dutch: Amazoneregenwoud), also known in English as Amazonia or the Amazon Jungle, is a moist broadleaf forest that covers most of the Amazon basin of South America. This basin encompasses 7,000,000 square kilometres (2,700,000 sq mi), of which 5,500,000 square kilometres (2,100,000 sq mi) are covered by the rainforest. This region includes territory belonging to nine nations. The majority of the forest is contained within Brazil, with 60% of the rainforest, followed by Peru with 13%, Colombia with 10%, and with minor amounts in Venezuela, Ecuador, Bolivia, Guyana, Suriname and French Guiana. States or departments in four nations contain "Amazonas" in their names. The Amazon represents over half of the planet's remaining rainforests, and comprises the largest and most biodiverse tract of tropical rainforest in the world, with an estimated 390 billion individual trees divided into 16,000 species.

Which name is also used to describe the Amazon rainforest in English?
Ground Truth Answers: also known in English as Amazonia or the Amazon Jungle, | Amazonia or the Amazon Jungle | Amazonia
Prediction: Amazonia

How many square kilometers of rainforest is covered in the basin?
Ground Truth Answers: 5,500,000 square kilometres (2,100,000 sq mi) are covered by the rainforest. | 5,500,000 | 5,500,000
Prediction: 5,500,000

How many nations control this region in total?
Ground Truth Answers: This region includes territory belonging to nine nations. | nine | nine
Prediction: nine

Source: SQuAD

column_cos_1	column_cos_2	column_cos_3	column_cos_4	column_cos_5	column_cos_6	column_cos_7	column_cos_8	column_cos_9	target
0.364050	0.347755	0.394242	0.371025	0.185690	0.351921	NaN	NaN	NaN	1

Encode the sentences (list of n sentences):

```
embeddings = infersent.encode(sentences, tokenize=True)
Next, Code the train_nli.py in the IDE:
import numpy as np
        import torch
        from torch.autograd import Variable
        import torch.nn as nn
        from data import get_nli, get_batch, build_vocab
        from mutils import get_optimizer
        from models import NLINet
        GLOVE_PATH = "dataset/GloVe/glove.840B.300d.txt"
        parser = argparse.ArgumentParser(description='NLI training')
        # paths
        parser.add_argument("--nlipath", type=str, default='dataset/SNLI/', help="NLI data path (SNLI
or MultiNLI)")
        parser.add_argument("--outputdir", type=str, default='savedir/', help="Output directory")
        parser.add_argument("--outputmodelname", type=str, default='model.pickle')
        # training
        parser.add_argument("--n_epochs", type=int, default=20)
        parser.add_argument("--batch_size", type=int, default=64)
        parser.add_argument("--dpout_model", type=float, default=0., help="encoder dropout")
        parser.add_argument("--dpout_fc", type=float, default=0., help="classifier dropout")
        parser.add_argument("--nonlinear_fc", type=float, default=0, help="use nonlinearity in fc")
        parser.add_argument("--optimizer", type=str, default="sgd,lr=0.1", help="adam or sgd,lr=0.1")
        parser.add_argument("--lrshrink", type=float, default=5, help="shrink factor for sgd")
        parser.add_argument("--decay", type=float, default=0.99, help="lr decay")
        parser.add_argument("--minlr", type=float, default=1e-5, help="minimum lr")
        parser.add_argument("--max_norm", type=float, default=5., help="max norm (grad clipping)")
```

```
        # model
        parser.add_argument("--encoder_type", type=str, default='BLSTMEncoder', help="see list of
encoders")
        parser.add_argument("--enc_lstm_dim", type=int, default=2048, help="encoder nhid dimension")
        parser.add_argument("--n_enc_layers", type=int, default=1, help="encoder num layers")
        parser.add_argument("--fc_dim", type=int, default=512, help="nhid of fc layers")
        parser.add_argument("--n_classes", type=int, default=3,
help="entailment/neutral/contradiction")
        parser.add_argument("--pool_type", type=str, default='max', help="max or mean")
np.random.seed(params.seed)
torch.manual_seed(params.seed)
torch.cuda.manual_seed(params.seed)
def evaluate(epoch, eval_type='valid', final_eval=False):
         nli_net.eval()
         correct = 0.
         global val_acc_best, lr, stop_training, adam_stop
         if eval_type == 'valid':
         print('nVALIDATION : Epoch {0}'.format(epoch))
         s1 = valid['s1'] if eval_type == 'valid' else test['s1']
         s2 = valid['s2'] if eval_type == 'valid' else test['s2']
         target = valid['label'] if eval_type == 'valid' else test['label']
         for i in range(0, len(s1), params.batch_size):
         # prepare batch
         s1_batch, s1_len = get_batch(s1[i:i + params.batch_size], word_vec)
         s2_batch, s2_len = get_batch(s2[i:i + params.batch_size], word_vec)
         s1_batch, s2_batch = Variable(s1_batch.cuda()), Variable(s2_batch.cuda())
         tgt_batch = Variable(torch.LongTensor(target[i:i + params.batch_size])).cuda()
         # model forward
         output = nli_net((s1_batch, s1_len), (s2_batch, s2_len))
         pred = output.data.max(1)[1]
         correct += pred.long().eq(tgt_batch.data.long()).cpu().sum()
         # save model
         eval_acc = round(100 * correct / len(s1), 2)
         if final_eval:
         print('finalgrep : accuracy {0} : {1}'.format(eval_type, eval_acc))
         else:
         print('togrep : results : epoch {0} ; mean accuracy {1} :
         {2}'.format(epoch, eval_type, eval_acc))
         if eval_type == 'valid' and epoch <= params.n_epochs:
         if eval_acc > val_acc_best:
         print('saving model at epoch {0}'.format(epoch))
         if not os.path.exists(params.outputdir):
         os.makedirs(params.outputdir)
         torch.save(nli_net, os.path.join(params.outputdir,
         params.outputmodelname))
         val_acc_best = eval_acc
         else:
         if 'sgd' in params.optimizer:
         optimizer.param_groups[0]['lr'] = optimizer.param_groups[0]['lr'] / params.lrshrink
         print('Shrinking lr by : {0}. New lr = {1}'
         .format(params.lrshrink,
         optimizer.param_groups[0]['lr']))
         if optimizer.param_groups[0]['lr'] < params.minlr:
         stop_training = True
         if 'adam' in params.optimizer:
         # early stopping (at 2nd decrease in accuracy)
         stop_training = adam_stop
         adam_stop = True
         return eval_acc
```

Develop a model based on natural language inference (SNLI):

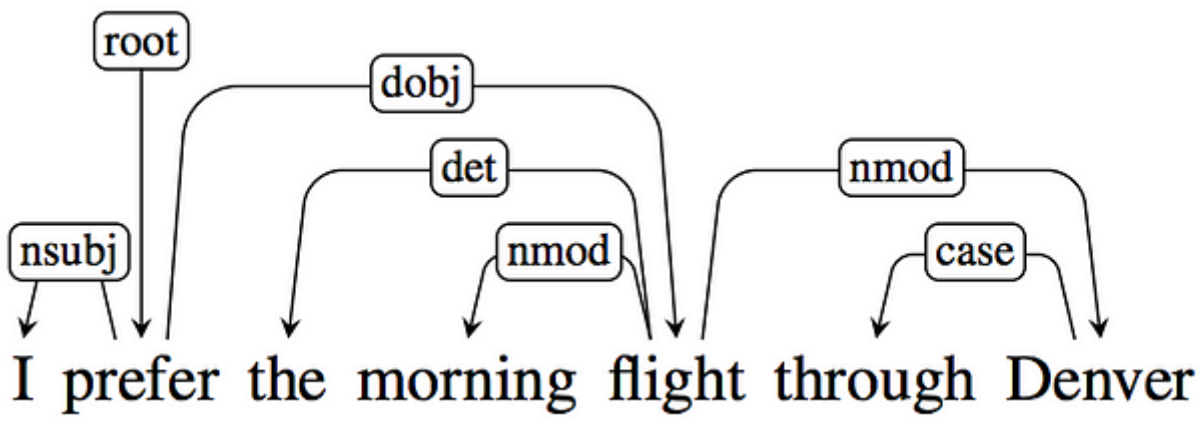
Set GLOVE PATH in train nli.py to replicate our results and train our models on SNLI, then run:

```
python train_nli.py
```

After the model has been trained, pass the sentence to the encoder function, which will produce a 4096-dimensional vector regardless of how many words are in the text.

Parsing Dependencies:

The “Dependency Parse Tree” is another feature I used to solve this problem. The model’s accuracy will improve by 5% because of this. Spacy tree parsing was used since it has a robust API for traversing through the tree.

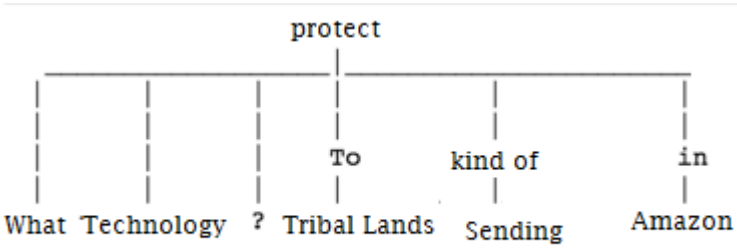


Above the text, directed, named arcs from heads to dependents show the relationships between the words. Because we generate the labels from a pre-defined inventory of grammatical relations, we call this a Typed Dependency structure. It also comprises a root node, which denotes the tree’s root, as well as the entire structure’s head.

Let’s use Spacy tree parse to show our data. I’m going to use the same example.

What kind of sending technology is being used to protect tribal lands in the Amazon?

```
[to_nltk_tree(sent.root) pretty_print() for sent in en_nlp(predicted.iloc[ 0, 2]).sents]
```



Sentence having the solution — *The use of remote sensing for the conservation of the Amazon is also being used by the indigenous tribes of the basin to protect their tribal lands from commercial interests.*

All Roots of the Sentences in the Paragraph are visualized:

```
for sent is doc.sents:
roots = [st.stem(chunk.root. head.text.lower()) for chunk in sent.noun_chunks ]
prlnt(roots)
```

```
['has', 'has']
['atop', 'is', 'of']
['in', 'of', 'fac', 'is', 'of', 'with', 'with', 'legend']
['to', 'is', 'of']
['behind', 'is', 'grotto', 'of', 'pray']
['is', 'is', 'of', 'at', 'lourd', 'appear', 'to']
['at', 'of', 'in', 'through', 'statu', 'is', 'of']
```

Lemmatization:

The Lemmatizer is a configurable pipeline component that supports lookup and rule-based lemmatization methods. As part of its language data, a language can expand the Lemmatizer.

Before comparing the roots of the sentence to the question root, it’s crucial to do stemming and lemmatization. Protect is the root word for the question in the previous example, while protected is the root word in the sentence. It will be impossible to match them unless you stem and lemmatize “protect” to a common phrase.

```
import spacy
nlp = spacy.load("en_core_web_sm")
lemmatizer = nlp.get_pipe("lemmatizer")
print(lemmatizer.mode)  # 'rule'
doc = nlp("The use of remote sensing for the conservation of the Amazon is also being used by the
indigenous tribes of the basin to protect their tribal lands from commercial interests.")
print([token.lemma_ for token in doc])
```

```
rule
['the', 'use', 'of', 'remote', 'sensing', 'for', 'the', 'conservation', 'of', 'the', 'Amazon', 'be', 'also', 'be', 'use', 'b
y', 'the', 'indigenous', 'tribe', 'of', 'the', 'basin', 'to', 'protect', 'their', 'tribal', 'land', 'from', 'commercial', 'in
terest', '.']
```

The goal is to match the root of the question, which in this case is “appear,” to all the sentence’s roots and sub-roots. We can gain several roots since there are multiple verbs in a sentence. If the root of the question is present in the roots of the statement, there is a better possibility that the sentence will answer the question. With this in mind, I’ve designed a feature for each sentence that has a value of 1 or 0. Here, 1 shows that the question’s root is contained in the sentence roots, and 0 shows that it is not.

We develop the transposed data with two observations from the processed training data model. So, for ten phrases in a paragraph, we have 20 characteristics combining cosine distance and root match. The range of the target variable is 0 to 9.

	0	1
column_root_0	0.000000	0.000000
column_root_1	0.000000	0.000000
column_root_2	0.000000	0.000000
column_root_3	0.000000	0.000000
column_root_4	0.000000	0.000000
column_root_5	1.000000	0.000000
column_root_6	0.000000	0.000000
column_root_7	0.000000	0.000000
column_root_8	0.000000	0.000000
column_root_9	0.000000	0.000000
column_cos_0	0.424736	0.454075
column_cos_1	0.364050	0.322620
column_cos_2	0.347755	0.355004
column_cos_3	0.394242	0.271561
column_cos_4	0.371025	0.392342
column_cos_5	0.185690	0.384383
column_cos_6	0.351921	0.362597
column_cos_7	1.000000	1.000000
column_cos_8	1.000000	1.000000
column_cos_9	1.000000	1.000000
target	5.000000	2.000000

This problem can also be solved using supervised learning, in which we fit multinomial logistic regression, random forest, and xgboost to construct 20 features, two of which represent the cosine distance and Euclidean distance for one sentence. As a result, we will limit each para to ten sentences).

```
import numpy as np, pandas as pd
import ast
from sklearn import linear_model
from sklearn import metrics
from sklearn.cross_validation import train_test_split
import warnings
warnings.filterwarnings('ignore')
import spacy
from nltk import Tree
en_nlp = spacy.load('en')
from nltk.stem.lancaster import LancasterStemmer
st = LancasterStemmer()
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
```

Load the dataset CSV file:

```
data = pd.read_csv("train_detect_sent.csv").reset_index(drop=True)
```

Let's retrieve the Abstract Syntax Tree for the DataFrames:

```
ast.literal_eval(data["sentences"][0])
```

After all, create a feature for the Data Frames and then train the model:

```
def crt_feature(data):
    train = pd.DataFrame()
    for k in range(len(data["euclidean_dis"])):
        dis = ast.literal_eval(data["euclidean_dis"][k])
        for i in range(len(dis)):
            train.loc[k, "column_euc_"+"%s"%i] = dis[i]
    print("Finished")
    for k in range(len(data["cosine_sim"])):
        dis = ast.literal_eval(data["cosine_sim"][k].replace("nan","1"))
        for i in range(len(dis)):
            train.loc[k, "column_cos_"+"%s"%i] = dis[i]
    train["target"] = data["target"]
    return train
```

```
train = crt_feature(data)
train.head(3).transpose()
```

Train the model using Multinomial Logistic regression:

```
train_x, test_x, train_y, test_y = train_test_split(X,
train.iloc[:,-1], train_size=0.8, random_state = 5)
```

```
mul_lr = linear_model.LogisticRegression(multi_class='multinomial', solver='newton-cg')
mul_lr.fit(train_x, train_y)
```

```
print("Multinomial Logistic regression Train Accuracy : ", metrics.accuracy_score(train_y,
mul_lr.predict(train_x)))
print("Multinomial Logistic regression Test Accuracy : ", metrics.accuracy_score(test_y,
mul_lr.predict(test_x)))
```

```
Multinomial Logistic regression Train Accuracy :  0.6373448858212791
Multinomial Logistic regression Test Accuracy :  0.6398026315789473
```

The phrase ID with the right answer is the target variable. As a result, I have ten labels. The accuracy is currently 63%, 65 per cent, and 69 per cent, respectively, on the validation set.

About Myself:

Hello, my name is Lavanya, and I'm from Chennai. I am a passionate writer and enthusiastic content maker. The most intractable problems always thrill me. I am currently pursuing my B. Tech in Computer Science Engineering and have a strong interest in the fields of data engineering, machine learning, data science, and artificial intelligence, and I am constantly looking for ways to integrate these fields with other disciplines such as science and chemistry to further my research goals.

Linkedin URL: https://www.linkedin.com/in/lavanya-srinivas-949b5a16a/

Quick Notes to Grasp:

1) Natural language processing (NLP) is an area of machine learning and artificial intelligence that is snowballing. Simply, machine learning is teaching computers to read, understand, and process human languages. We can build hundreds of applications in an NLP project, including search, spell check, auto-correct, chatbots, product suggestions, and more.

2) Natural language processing is used to answer questions, which is a simple and common task. A natural language processing system can match a user's query to your question and present the most relevant answer automatically. We called this as 'Automated question answering'

I hope this blog will be more informative and interesting!

If you have questions, please leave them in the comments area. In the meantime, check out my other articles here!

Thank you for reading.

The media shown in this article is not owned by Analytics Vidhya and are used at the Author's discretion

blogathon Question-Answering System

About the Author

LAVANYA S

Hello, my name is Lavanya, and I'm from Chennai. I am a passionate writer and enthusiastic content maker. The most intractable problems always thrill me. I am currently pursuing my B. Tech in Computer Engineering and have a strong interest in the fields of data engineering, machine learning, data science, and artificial intelligence, and I am constantly looking for ways to integrate these fields with other disciplines such as science and computer to take further my research goals.

Our Top Authors

Download

Analytics Vidhya App for the Latest blog/Article

Previous Post

Artificial Neural Network and Its Implementation From Scratch

Next Post

A Complete Beginner-Friendly Guide to SQL for Data Science

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

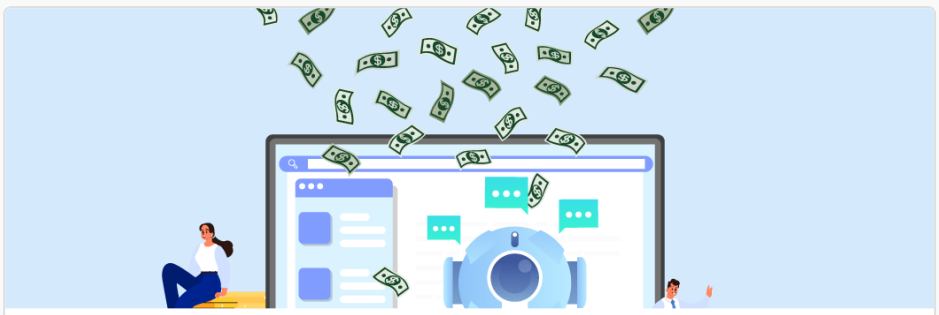
Website

Notify me of follow-up comments by email.

Notify me of new posts by email.

Submit

Top Resources



From Zero to Millionaire: Generate Passive Income using ChatGPT

Aravindpai Pai - APR 17, 2023

Microsoft Releases VisualGPT: Combines Language and Visuals

K.sabreena - APR 15, 2023

FreedomGPT: Personal, Bold and Uncensored Chatbot Running Locally on Your..

K.sabreena - APR 08, 2023

Meet AgentGPT, an AI That Can Create Chatbots, Automate Things,..

Sakshi Khanna - APR 16, 2023

Post Jobs

Trainings

Hiring Hackathons

Advertising