NLP PAPER REPORT

TEAM WORK

YAMOAH BENTIL, NANIS KANANA, SEYRAM KARTEY, FAITH KILONZI, WAYNE GAKUO

NLP Project, WRITE UP

Wayne Gakuo, Yamoah Bentil, Seyram Kartey, Nanis Kanana, Faith Kilonzi

In this project, we implemented models for question answering and topic modelling. Some of the approaches used are logistic regression, naïve Bayes, minimum edit distance, cosine similarity and Viterbi algorithms. The minimum edit distance model was best in performance so the final solution is based on it.

Topic Modelling

#concept 1: Logistic Regression for Topic Modelling

We used multinomial Logistic Regression for topic modeling. The classifier takes a bag of words. During training, we generate a tf-idf matrix representation of the data that is used for training. Because I am setting "topics" as the classes for the model, this is a multiclass classification; classification task with more than two classes; e.g., Government of Ghana, Networking, Blockchain, Common IT issues etc. Multiclass classification assumes that each sample is assigned to one and only one label which means a question can only be assigned to one topic.

The classifier is trained on the Questions and Topics datasets such that given a new dataset of questions it can easily assign the most probable topic. After loading the datasets, we cleaned it by tokenization; split the text of input documents into individual tokens, each corresponds to a single term of the document. Then we did lemmatization and stemming (removing tense or plurals from the word as same word can be used with different tenses/plurals e.g. computer, computing). When tokenization is done, we created a bag of words, every document is represented as term vector along with the number of times a term appeared in that document. In text processing, we also did stop words filtering (removing terms that do not convey any meaning to topic e.g.—at, the, as, an etc.). We then used TF-IDF for term weighting. We then used sklearn library for training by first splitting data into training and test sets. Training the model on the data, storing the information is learned from the data. The model learns the relationship between questions (x_train) and topics (y_train) and then we use the information the model learned during the model training process to predict topics for new questions. We used accuracy score model to measure model performance. The model gave an accuracy of 65.04%. We also used pandas for data processing.

#**Concept 2: Using Naïve Bayes for Topic Modelling**

We built a naïve Bayes classifier to model topics given a series of questions. To represent the documents, we used a TF-IDF vector which helps in modelling text into vectors and mathematical models. For data preprocessing, we used genism library along with nltk for faster and efficient data preprocessing. We then trained and built the model using Naïve Bayes then predicted the output with some test data. The model accuracy was also evaluated. The main idea is to train a pair of questions-answer texts, after which the model learns from it then it can accept new questions to model their respective topics.

The Naïve Bayes model was built on the *sklearn* Naïve Bayes *MultinomialNB* classifier. This is because the multinomial classifier is useful when processing sparse data through word counts and frequency. Other libraries used in the program include *nltk*, *numpy* and *pandas* for text reading, feature extraction, preprocessing, vectorization and visualization. Python's *io* library was also imported for file processing.

#Concept 3: Using Minimum Edit Distance for Topic Modelling and Question Answering

Minimum edit distance is an algorithm used to find the number of edits it takes to change one string to another. In this approach, we look at the text similarity of sentences given a text corpus calculating the Euclidean distance between two sentences.

Cleaning the data: The training data that was given had too many special Unicode blocks, inconsistent numbering and tab spaces. So, to make all the data uniform. I needed to create a function to clean the data using regular expressions and python string stripping functionality.

Reading the data: The algorithm takes the files from the three different test files, Questions, Answers and Topics and then cleans all of them using the clean data function to make the files consistent. We then append all the files to lists that act as a reference to their corpuses. Thus, we have the questions corpus, answer corpus and topic corpus lists.

Training the data and determining the closest questions: A function called *checkClosestDistance* handles the Euclidean computations to determine how similar the texts are by determining the text features that can be replaced and are similar to each. So firstly, the algorithm uses *sklearn* to strip the sentences off things like punctuations, Capitals, suffixes etc. Then features just transforms the data to fit and suit the corpus that we have generated. We then insert the test data as part of the corpus then we loop through the entire data set checking to see which sentence matches with test data. When complete, the smallest distances are considered the closest to the sentence of interest. We then check for the index of the smallest distance and checks the question that is the nearest to the users' question asked and return the index for the answering and topic modelling

The answering and topic model: The answer and topic model basically work the same way, when it receives an index, and the sentence being searched. Then, it checks the corresponding indices that belong to the Topic and Answer corpuses then returns the values contained at that index and prints the results to the file.

Question Answering

#1: Cosine Similarity

In this approach, we look at how we can answer user's questions using cosine similarity. This is done by comparing the user's questions and the model's trained questions. This is mostly used when one needs to carry out sentimental analysis, translation or simply building a chatbot that answers user's questions on a certain topic. The main idea of using cosine similarity is to obtain a

theme from a text without worrying about the length of text. Cosine similarity is determined by finding how close two vectors (each representing a sentence) are: if the vectors are parallel, we can infer that the sentences are similar. If the vector is orthogonal, we might infer that the sentences are independent of each other.

Cleaning Data: Based on the training data, we tokenize the content by splitting up the sentences into words, putting all of them to lower cases, removing non-alphabetical characters, removing stop-words and whitespaces. The cleaning of the data is done to remove unnecessary data which do not provide much significance in our training data. We also remove the numbers from the beginning of the lines to be able to only work with the words since the numbers are used as a numbering system.

Creating a bag of words: We do this by creating a set where each word will be unique based on the training data. This creates a 'bag of words' which we will use in training the model. The bag of words will be used as a reference for the training of the model and testing.

Training: On each line of the test data, we check for the occurrence of a word from the bag of words. If found, a value of 1 is added into an array, else, zero is added. This is done for every line and a general array is created to hold arrays for the different lines. Therefore, each line will contain an array denoted with zero's and one's and each array serves as an input to the general array (an array of arrays). Words (in this case, the arrays) are then compared against each other to find which ones are almost similar. The pair with the highest probability (closer to 1) is chosen and the answer to corresponding to the training question from the "Answers" text file is returned to the user.

Testing: In testing, the text file is passed through the cosine-similarity model and based on the user's question, the appropriate answer is provided.

#2 Using Minimum Edit Distance for Question Answering

(illustration explained above)

#3 POS Tagging: The Viterbi algorithm can be used here for the purposes of POS tagging.

The algorithm to be used here can behavior like the Viterbi algorithm. Since we are going to train the algorithm on a labelled dataset, this is how we train the algorithm. Since we are going to provide the algorithm with labelled dataset, it can be taught of as supervised machine learning.

Breakdown: The topics we have can thought of as the states (the same way the Parts of speech are represented when using the Viterbi algorithm). Each state is like bag of words, keeping track and count of the words that are stored in it. Or instead of keeping a separate "bag of words", we can have just one, and instead store words as objects that have an attribute state.

To make the system more efficient more accurate, we are going to insist on trigrams as the minimal check. Thus, words are going to be compared in at least orders of three. We will be using strict trigrams. In more complex systems, the n-gram can be increased to further enhance the accuracy of the system.

Each question and answer pair within each topic are connected. If a user types a question, the answer that will be given will the linked answer of the question that best matches the users question.

Given a question, iterates through the sentence and compares the trigrams with that of the various states from the trained data.