

MY FIRST CHAT BOT :PYTHON
PROJECT

MERMONT--DOMINGUEZ SOPHIE
YOUSAME MBILI JOY
L1INT4-2023

|

--Table of contents--

1.	<u>Introduction</u>	<u>3</u>
2.	<u>Functional Presentation</u>	<u>3</u>
3.	<u>Technical Presentation</u>	<u>4</u>
4.	<u>Difficulties Faced</u>	<u>5</u>
5.	<u>Result presentation</u>	<u>6</u>
6.	<u>Conclusion</u>	<u>7</u>

--Introduction to the projet--

This project explores text analysis concepts for creating a question-answering system. It employs a method based on word frequency to generate intelligent responses from a text corpus. The algorithm involves data pre-processing, where documents are cleaned and tokenised. A TF-IDF matrix is constructed, associating each word with a vector reflecting its occurrence across documents. When a question is asked, the chatbot calculates a TF-IDF vector and assesses its similarity to word vectors in the corpus. The system then selects the best answer based on TF-IDF similarity scores, considering words most similar to the question. Ultimately, the chatbot provides the selected answer as the response. The focus is on designing a system adept at answering questions by leveraging word frequency in the corpus.

--Functional Presentation--

- 1.**extract_president_names(file_names)**:Returns a list of the names of each president from the speech file names.
- 2.**clean_text(path, speech)**:Converts the speeches to lowercase and removes punctuation.
- 3.**count_word_occurrences(text)**:Counts the number of times a word is present in a text.
- 4.**calculate_tf(path)**:Returns the term frequency of each word in a text.
- 5.**in_doc(directory)**:Returns a dictionary with the number of speeches each word is present in, used in calculate_idf().
- 6.**calculate_idf(path)**:Returns a dictionary with the IDF score of each word for a given speech.
- 7.**calculate_tf_idf_matrix_all(directory)**:Returns a TF-IDF matrix for all speeches in the directory.
- 8.**display_least_important_words(tf_idf_matrix)**:Displays the list of least important words.
- 9.**display_highest_tfidf_words(tf_idf_matrix)**:Displays word(s) with the highest TF-IDF score.
- 10.**calculate_tf_idf_matrix_with_presidents(directory, president)**:Returns a TF-IDF matrix of each speech for a given president in a given directory.
- 11.**most_repeated_words_by_president(directory, president)**:Returns the most repeated word said by a given president.
- 12.**word_frequence_comparison(directory, target_word)**:Returns a list of who says the target word more in its speeches.
- 13.**calculate_tf_idf_matrix_with_target_word(directory, president, target_word)**:Returns a TF-IDF matrix of each speech for a given president with respect to a target word.
- 14.**president_speaking_about_nation_the_most(tf_idf_matrix, president_name)**:Indicates the most repeated word(s) by a given president.
- 15.**word_frequence_comparison(directory, target_word)**:Returns a list of who says the target word more in its speeches.
- 16.**calculate_tf_idf_matrix_with_first_mention(directory, president, target_words)**:Returns a TF-IDF matrix with the first mention of target words.
- 17.**first_president_to_mention_climate_or_ecology(tf_idf_matrix, target_words)**:Identifies the first president to talk about 'Climate' and/or 'Ecology'.
- 18.**words_mentioned_by_all_presidents(tf_idf_matrix, unimportant_words)**:Returns words mentioned by all presidents, excluding unimportant words.
19. **main program**:that provides a menu to interact with different functionalities.
- 20.**question_token(question)**:Tokenizes a question, returning a cleaned version as a list of words.
- 21.**allcorpus(directory)**:Returns all the speeches as a single string.
- 22.**tokens_in_corpus(token_list, directory)**:Returns a list of tokens that are in the corpus.
- 23.**tf_idf_matrix_fix_separated_by_document(directory)**:Returns a TF-IDF matrix that depends on each independent document.

--Technical presentation--

1. Data Pre-processing:

--Text cleaning involves removing punctuation and converting letters to lowercase.

--The text is split into words or tokens.

This step ensures a standardized format for subsequent analysis.

2. Creating a TF-IDF Matrix:

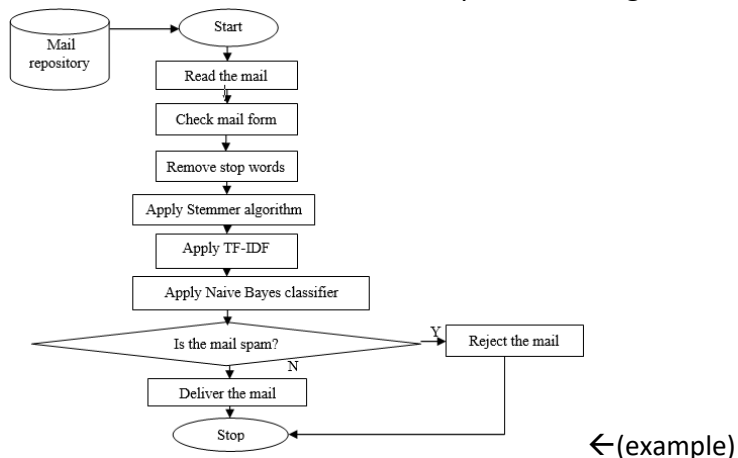
--For each unique word, calculate Term Frequency (TF) and Inverse Document Frequency (IDF).

--Create a TF-IDF vector for each word, resulting in a matrix where rows represent words, and columns represent documents.

3. Question Representation:

--Pre-process the user's question similarly to the data, ensuring consistency.

--Calculate a TF-IDF vector for the question using the same vocabulary as the documents.



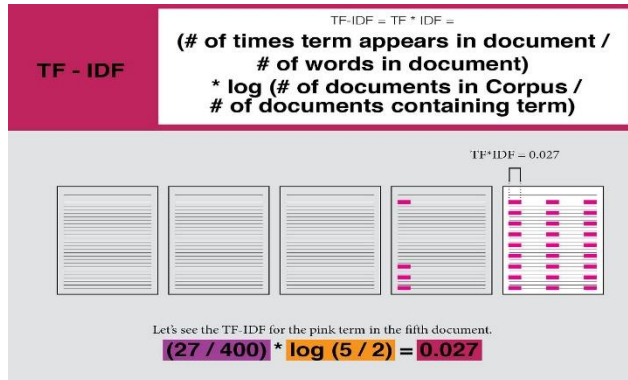
4. Similarity Calculation:

--Use cosine similarity or another measure to quantify the similarity between the question vector and word vectors in the corpus.(This step identifies relevant words in the corpus.)

5. Selecting the Best Answer:

--Based on TF-IDF similarity scores, determine words most similar to the question.

--Select the answer with the highest frequency of these similar words.

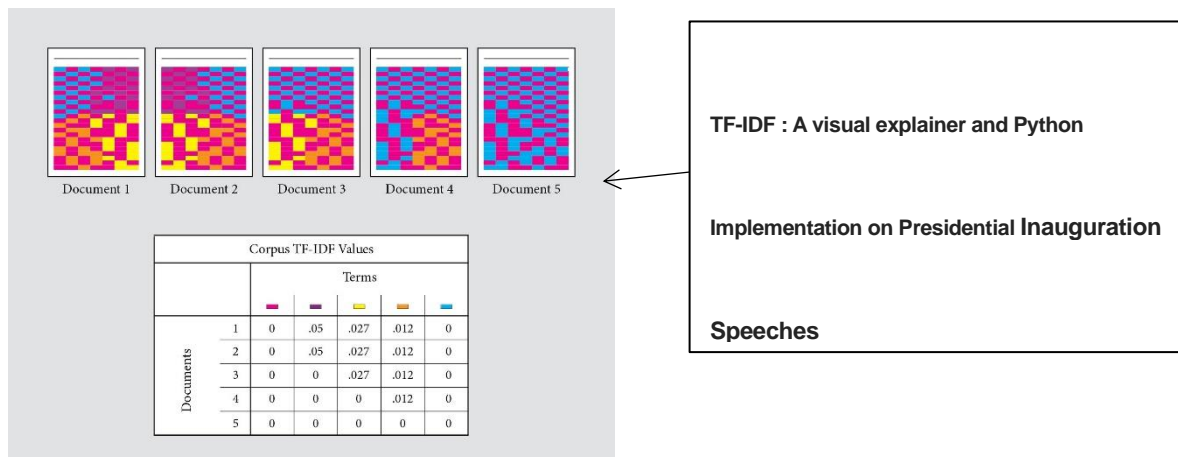


6. Text Analysis and TF-IDF Calculation:

--Calculate TF for each word's frequency in a document.

--Determine IDF, measuring a word's importance across multiple documents.

--Combine TF and IDF to obtain the TF-IDF score for each word in each document.



Difficulties Faced and solutions created:

=We had a dictionary system for the tf-idf.

- We remade the tf-idf matrix as a proper matrix.

=We had functions that didn't work.

- We made updated versions of them but now we have multiple function that do the same thing but some of them don't work.

=We had a problem with our commit and the one team member couldn't push all the functions that they made.

-We created a word document "CommitsThatJoyYousameHasDone" allowing the other member to make their own function and all we need to do is Ctrl+C on PyCharm.

Results presentation:

The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for file operations, running, and debugging. The left sidebar displays the project structure, with the 'main.py' file selected. The main editor window shows the following Python code:

```

541
542 # Function 1
543 3 usages 1 smiermont-ehfi
544 def question_token(question):
545     """Takes a question as a string in parameter and returns a cleaned version of it as a list of words (lowercase and with no punctuation)"""
546     question = str(question)
547     text = question # I put the question in lowercase manually after taking away the punctuation
548
549     # We take away the punctuation
550     punctuation = ('!', '"', '#', '$', '%', '&', '\'', '*', '+', ',', '-', '.', ':', ';', '?', '@', '^', '_', '`', '~', '{', '}', '|', '~', '~')
551     text1 = ''
552     for word in text:
553         for char in word:
554             if char not in punctuation:
555                 text1 += char
556             else:
557                 text1 += ' '
558
559     text = text1
560
561     # We turn letters into lowercase when needed
562     text1 = ''
563     for word in text:
564         for letter in word:
565             if 64 < ord(letter) and ord(letter) < 91 :
566                 text1 += chr(ord(letter) + 32)
567             else:
568                 text1 += letter
569
570     text = text1
571     lquestion = text.split()
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

The bottom status bar shows the current file is 'main.py' and the encoding is 'UTF-8'.

The screenshot shows the VS Code interface with the following components:

- Top Bar:** Displays the file name `main.py`, the current branch `master`, and icons for file operations and search.
- Left Panel:**
 - Commit View:** Shows a list of commits. The selected commit is `main.py` with a message `Function 5 part 2 doesn't work`. It indicates `1 modified` file.
 - Run Panel:** Shows the command `main` being executed.
- Editor:** Displays the Python code for `main.py`. The code includes a function `test_function` that processes a string and a test call. Line 572 is highlighted in blue.


```

570
571 # Testing the function :
572 print(question_token('Comment s'appelle le président ?'))
573

```
- Output Panel:** Shows the execution output:


```

"C:\Users\samier\OneDrive\Documents\Saphir\Scolaire\Efre1 2023 2024\Cours\Semestre 1\Python\Projet chatbot\pychatbot-Miermont-Dominguez-Youssame-int4\venv\Scripts\python.exe"
['comment', 's', 'appelle', 'le', 'président']

Process finished with exit code 0

```
- Bottom Bar:** Shows the file explorer with `pychatbot-Miermont-Dominguez-Youssame-int4` and `main.py`.

--Conclusion--

The project has been a valuable learning experience on multiple fronts. From a technical perspective, we have deepened our understanding of text processing, TF-IDF algorithms, and practical applications of natural language processing. Collaborating as a duo highlighted the importance of effective communication and task coordination. Time management skills were very important while, problem-solving became a daily practice as we encountered and overcame technical challenges. Feedback integration played a crucial role in refining the project and improving team performance. Overall, the project provided a comprehensive platform for technical development.