

Assignment 1C: Medical Diagnosis App

Anupam Kumar(11940160)

- Dataset Preparation

- Data Collection:

The dataset used is taken from Kaggle:

<https://www.kaggle.com/itachi9604/disease-symptom-description-datase>.

But since this dataset was not sufficient so, we collected data for diseases from the [National Health Portal](#) of India which provides a wide variety of diseases in the human body. The symptoms have been scraped from Wikipedia using the **googlesearch** library.

The data is scraped by targeting the correct HTML tags.

```
# Fetch disease list from 'www.nhp.gov.in'
small_alpha = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's',
diseases=[]
for c in small_alpha:
    URL = 'https://www.nhp.gov.in/disease-a-z/'+c
    time.sleep(1)
    page = requests.get(URL,verify=False)

    soup = BeautifulSoup(page.content, 'html5lib')
    all_diseases = soup.find('div', class_='all-disease')

    for element in all_diseases.find_all('li'):
        diseases.append(element.get_text().strip())

with open('list_diseaseNames.pkl', 'rb') as handle:
    diseases2 = pickle.load(handle)
```

○ Data Cleaning

After the raw data was scraped, the data was cleaned by removing stop words, converting the symptoms into lowercase, and splitting it based on commas. After that, synonymous symptoms or diseases(if encountered) were also omitted which was done by using the Wordnet library and thesaurus.

```
# Go over all disease and preprocess symptoms string and break it into individual symptoms
for key in sorted(dis_symp.keys()):
    value = dis_symp[key]
    list_sym = re.sub(r"\\s+", "", value).lower().split(',')
    temp_sym = list_sym
    list_sym = []
    for sym in temp_sym:
        if len(sym.strip())>0:
            list_sym.append(sym.strip())
    # Remove 'none' from symptom
    if "none" in list_sym:
        list_sym.remove("none");
    if len(list_sym)==0:
        continue
    temp = list()
    for sym in list_sym:
        sym=sym.replace('-', ' ')
        sym=sym.replace("'", '')
        sym=sym.replace('(', '')
        sym=sym.replace(')', '')
        sym = ' '.join([lemmatizer.lemmatize(word) for word in splitter.tokenize(sym) if word not in stop_words and not word[0].is
        total_symptoms.add(sym)
        temp.append(sym)
    diseases_symptoms_cleaned[key] = temp
```

```

# stores the synonym for each symptom in the list of words
sym_syn = dict()
for s in total_symptoms:
    symp=s.split()
    str_sym=set()
    for comb in range(1, len(symp)+1):
        for subset in combinations(symp, comb):
            subset=' '.join(subset)
            subset = synonyms(subset)
            str_sym.update(subset)
    str_sym.add(s)
    str_sym = ' '.join(str_sym).replace('_', ' ').lower()
    str_sym = list(set(str_sym.split()))
    str_sym.sort()
    sym_syn[s] = str_sym

```

```

# Iterate over all the symptoms in dataset and check the similarity score to the synonym string of the user-input symptoms
# Now, If similarity>0.5, add the symptom to the final list
found_symptoms = set()
for idx, data_sym in enumerate(dataset_symptoms):
    data_sym_split=data_sym.split()
    for user_sym in user_symptoms:
        count=0
        for symp in data_sym_split:
            if symp in user_sym.split():
                count+=1
        if count/len(data_sym_split)>0.5:
            found_symptoms.add(data_sym)
found_symptoms = list(found_symptoms)

```

○ Data Processing

The data is finally converted to binary vectors which have a value of 1 if some disease has that index of symptom(s).

★ The final data frame obtained has been attached in the zipped file.

- **Training the Decision Tree**

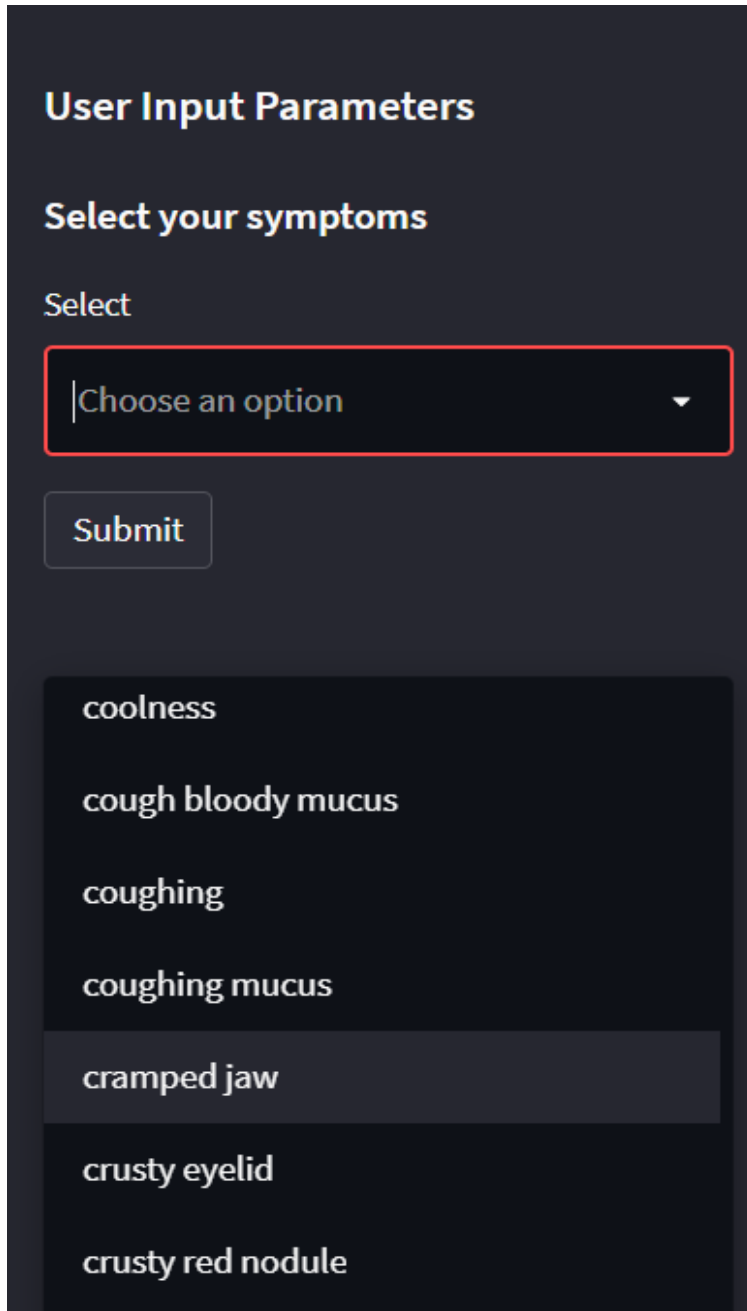
The decision tree was trained using the `DecisionTreeClassifier` from the `sklearn` library. The accuracy obtained is 90.05% whereas the Cross-Validation Accuracy is around 84%.

The decision tree is also implemented by calculating the total entropy and getting the information gained at each step to decide the next root node and then getting the leaf nodes at the end with probabilities that represent the disease at the end.

- **Interactive Web Application**

- [Streamlit](#) has been used to create the Web Application using Python for the ML model.

- The user is prompted to select the sympto(s) from the list of symptoms from the drop-down menu on the left side of the web page as shown.



User Input Parameters

Select your symptoms

Select

Choose an option ▼

Submit

- coolness
- cough bloody mucus
- coughing
- coughing mucus
- cramped jaw
- crusty eyelid
- crusty red nodule

- The user can select multiple inputs at the same time and as the Submit button is pressed the event triggers the predict_diseases() function which takes in the entries in the symptoms which have been selected.
 - Inside the function, we generate the binary vector corresponding to the total available symptoms, and then this is used further to classify using the decision tree. The code for this can be found in the zipped file as well.
 - After the model has been successfully executed in the backend, the symptoms selected by the user are displayed first followed by the list of diseases and their chances of occurrence(in %) sorted in decreasing order with respect to the probability obtained.
- ★ The code to run this software has been attached in the zipped file.

Medical Diagnosis App

Predicting the disease based on the symptoms

Please enter the symptoms you are experiencing in the sidebar.

You selected the following symptoms:

▼ [

```
0 : "asymptomatic early stage"  
1 : "bad breath"  
2 : "barking cough"  
3 : "black area skin"  
4 : "bloating"  
5 : "numbness"
```

]

Disease Prediction by the Decision Tree Classifier

The top 10 results from the model along with their corresponding chances of occurrence(in %) are:

```
▼ [
  ▼ 0 : [
    0 : "Abnormal uterine bleeding"
    1 : 26.3
  ]
  ▼ 1 : [
    0 : "Acquired Capillary Haemangioma of Eyelid"
    1 : 26.3
  ]
  ▼ 2 : [
    0 : "Acquired Immuno Deficiency Syndrome"
    1 : 13.15
  ]
  ▼ 3 : [
    0 : "Acute encephalitis syndrome"
    1 : 13.15
  ]
  ▼ 4 : [
    0 : "Adult Inclusion Conjunctivitis"
    1 : 13.15
  ]
]
```

● Steps to run the code

- Directly run the medical_diagnosis.py file after having fulfilled the requirements of the libraries needed.
- The Streamlit automatically returns the command to run the Web Application in your local machine.
- So, run that command and it redirects you automatically to the webpage.

● Further Improvements

- We have implemented the code part of scraping the data for the specific disease and displaying several information related to the diseases that have been resulted by the model like showing the treatment of the disease(if possible), some other common symptoms of the disease, etc.
- This has also been done by scraping data mostly from Wikipedia by targeting the HTML tags and using various inbuilt libraries.
- But, this part could not be implemented in the web application as some issues were encountered while trying to use Streamlit for this purpose.
- We can also provide the user to enter some symptoms which have not been mentioned in the dataset and the model would add that symptom if found unique and use that to get the diseases leading to it.
- A separate model could also be included which would predict the next steps in order to solve some disease be it suggesting some basic home remedy or suggesting the patient to immediately seek doctor's support based on the seriousness of his/her condition which could again be known by asking further questions from the patient.
- Another improvement would be to make the application more user-friendly which requires using some web development tools like React for front and Django for a backend which would be hosted permanently on some stable server.