

**AI in Health care**

**LAB FILE**

**Bachelor OF Technology**

**(Academic Session: 2019-2023)**

**Name of Student:** Divvela Vishnu Sai Kumar

**Enrollment No. :** A076119819005

**Branch:** B.Tech (AI) 7th Sem

**Course** **Name**: AI in Health care

**Course Code**: CSE462

**Faculty Name:** Prof. Dr. Pooja Khanna

**Faculty** **Name**: Dr Pooja Khanna

1

**Department of Computer Science & Engineering**

**Amity School of Engineering & Technology**

**Amity University, Uttar Pradesh**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO.** | **Program Title** | **Date** | **Remarks** |
| 1 | Build and train deep neural networks, and apply it to Health Care Data. |  |  |
| 2 | Implement Hidden Markov Model using Health Monitoring Activities dataset. |  |  |
| 3 | Topological Modelling and Classification Model of Health Care Data |  |  |
| 4 | Build a NLP Model to Recommend Healthcare Practitioners based on Patient Reviews |  |  |
| 5 | Simulation of Interpretability Machine Learning Algorithm |  |  |

**PRACTICAL-WORK**

Practical-1

OBJECTIVE:

Build and train deep neural networks, and apply it to Health Care Data.

Tools Used:

* Python
* Vs code editor
* Jupyter Notebook
* Tensorflow

Description:

In this practical we will choose the cancer dataset and we will try to classify the disease label using tensorflow deep neural network. We will classify the cancer into Benign and Malignant, Benign means no cancer and Malignant means there are chances to get cancer.

Code:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

data = pd.read\_csv('cancer\_classification.csv')

data.head(20)

sns.countplot(x='benign\_0\_\_mal\_1',data=data)

x = data.drop('benign\_0\_\_mal\_1',axis=1).values

y = data['benign\_0\_\_mal\_1'].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.33, random\_state=42)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Activation,Dropout

model = Sequential() #this will create a empty stack where we can start adding the layers in sequence

model.add(Dense(units=30,activation='relu'))

model.add(Dense(units=15,activation='relu'))

model.add(Dense(units=1,activation='sigmoid'))

model.compile(loss='binary\_crossentropy', from tensorflow.keras.callbacks import EarlyStopping

early\_stop = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=25)

model.fit(x=X\_train,

          y=y\_train,

          epochs=600,

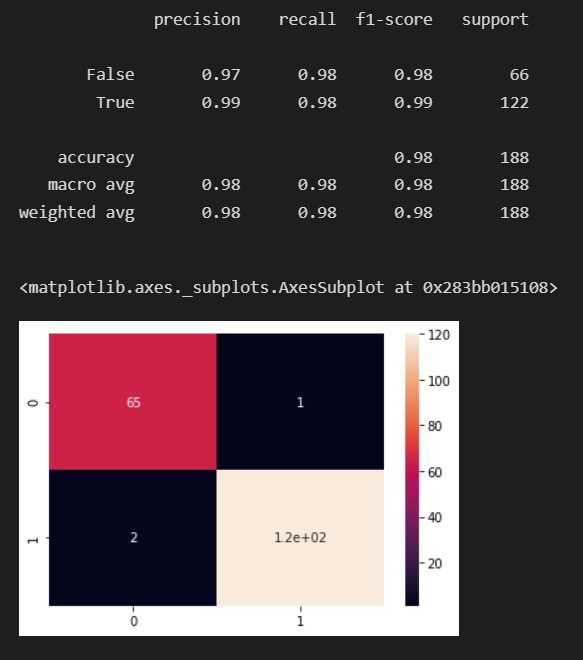
          validation\_data=(X\_test, y\_test), verbose=1,

          callbacks=[early\_stop]

          )

optimizer='adam',metrics=['accuracy'])

Output:



Practical-2

OBJECTIVE:

Implement Hidden Markov Model using Health Monitoring Activities dataset.

Tools Used:

* Python
* Vs code editor
* Jupyter Notebook
* Tensorflow

Description:

A Markov chain is a model that tells us something about the probabilities of sequences of random variables, states, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, like the weather. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state.

More formally, consider a sequence of state variables q1,q2,...,qi . A Markov model embodies the Markov assumption on the probabilities of this sequence: that Markov assumption when predicting the future, the past doesn’t matter, only the present.

Markov Assumption: P(qi = a|q1...qi−1) = P(qi = a|qi−1)

A Markov chain is useful when we need to compute a probability for a sequence of observable events. In many cases, however, the events we are interested in are hidden hidden: we don’t observe them directly. For example we don’t normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence.

We call the tags hidden because they are not observed. A hidden Markov model (HMM) allows us to talk about both observed events Hidden Markov model (like words that we see in the input) and hidden events (like part-of-speech tags) that we think of as causal factors in our probabilistic model.

Practical-3

OBJECTIVE:

Topological Modelling and Classification Model of Health Care Data

Tools Used:

* Python
* Vs code editor
* Jupyter Notebook
* Tensorflow

Description:

We often use machine learning to try to uncover patterns in data. In order for those patterns to be useful they should be meaningful and express some underlying structure. Geometry deals with such structure, and in machine learning we especially leverage local geometry. This can be seen in the Euclidean-inspired loss functions we use for generative models as well as for regularization. However, global geometry, which is the focus of Topology, also deals with meaningful structure, the only difference being that the structure is global instead of local. Topology is at present less exploited in machine learning, which is also why it is important to make it more available to the machine learning community at large.

Still, topology applied to real world data using persistent homology has started to find applications within machine learning (including deep learning), but again, compared to its sibling local geometry, it is heavily underrepresented in these domains. In this post, we provide a high-level description of how our TopologyLayer allows (in just a few lines of PyTorch) for backpropagation through Persistent Homology computations and provides instructive, novel, and useful applications within machine learning and deep learning.

As a teaser, consider Figure 1 below. We will show how, in just a few lines of code and a few iterations of SGD, we can define a topology loss and make a generator go from outputting images such as those on the left hand side to those on the right hand side, improving the topological fidelity.

Practical-4

OBJECTIVE:

Build a NLP Model to Recommend Healthcare Practitioners based on Patient Reviews

Tools Used:

* Python
* Vs code editor
* Jupyter Notebook
* Tensorflow

Description:

we are considering a dataset on the reviews of drugs by patients that contains features drugname, condition, review(text), rating, date, usefulcount. Usually the review given in the text format and rating in the number format wont match for example some patient can write the text as the medicine is very good and he gives rating 2 out of 10 or a user writes nothing in the text and he gives 9 out of 10 rating, so in these cases that review is not useful, so we want to help the healthcare pratitioner with good reviews for example if the patient given the text review very badly then he should give the rating also very less, and if patient given the text review as the medicine is very good and it cured my disease then he should give the rating atleast 9 out of 10.

So for this problem statement first we built a neural network classifier for text data which classifies the given text as positive review or negative review with a probability of prediction (this neural network is trained on imdb review dataset that can be found in tfds). Now using that model, we will predict the probability of positive ness of patient text review and we will difference this value with numerical rating value. So if it is less than 15% then the text review and the rating given by the patient are identical so we will classify that whole review as genuine review(1) otherwise we will classify that whole review as non-genuine review (0).

Code:

import numpy as np

import pandas as pd

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

import tensorflow as tf

data = pd.read\_csv('drugsComTrain\_raw.tsv',sep="\t")

data = data.iloc[:1000,:]

data.drop(["Unnamed: 0","date"],axis=1,inplace=True)

import pickle

import keras

loadedtokenizer = pickle.load(open('imdb\_tokenizer.pkl','rb'))

loadedmodel= keras.models.load\_model('imdb\_review\_feedback\_emotion.h5')

def positivity\_score(text):

    data4 = [text]

    data4=loadedtokenizer.texts\_to\_sequences(data4)

    data4=pad\_sequences(data4,maxlen=120,truncating='post')

    prediction4 = loadedmodel.predict(data4)[0][0]

    prediction4 = int(prediction4\*100)

    return prediction4

data["positivity\_score"]=data["review"].apply(positivity\_score)

def change\_rating(num):

    num = int(num\*10)

    return num

data['rating']=data['rating'].apply(change\_rating)

def genuinity\_calculator(rating,positivity\_score):

    diff = rating-positivity\_score

    if diff<0:

        diff = -1\*diff

    #if review and rating is almost same then 1 is label else 0 is label, for this the threshold is it should be atleast 15% similar

    if diff<=15:

        return 1

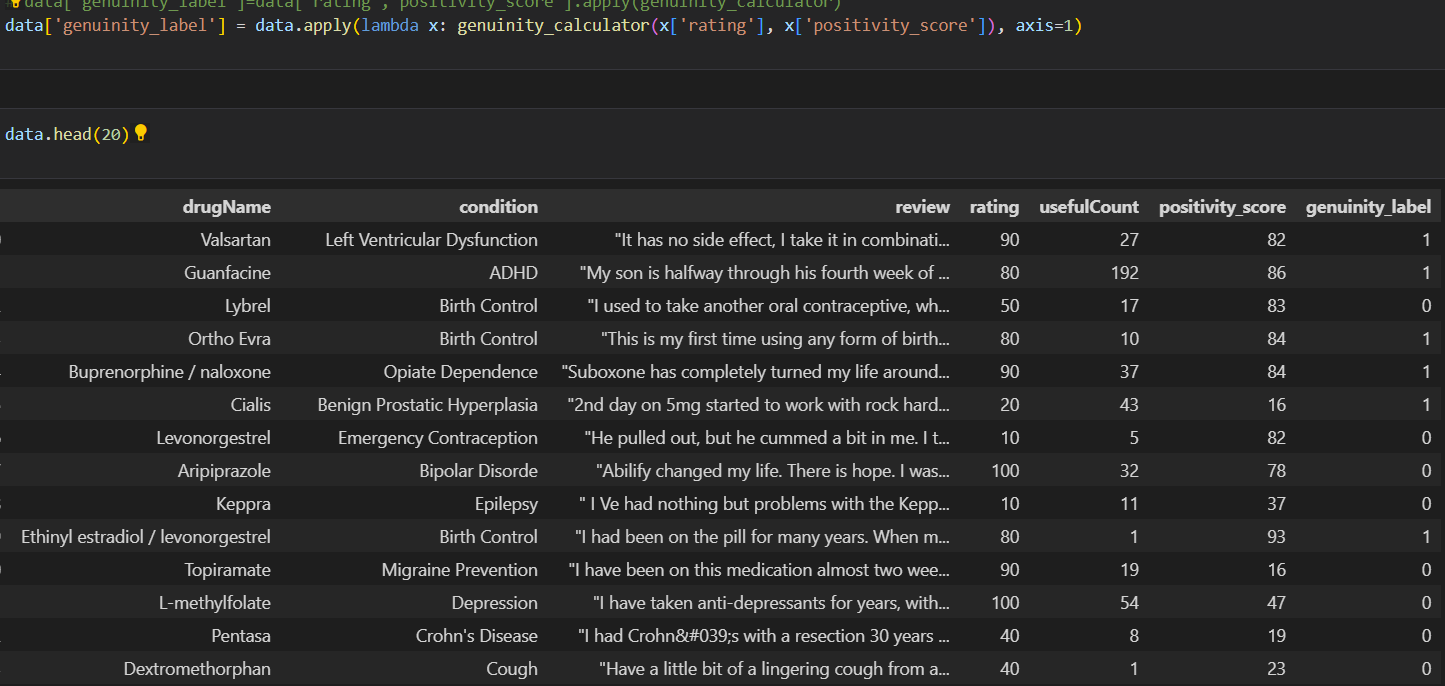
    else:

        return 0

data['genuinity\_label'] = data.apply(lambda x: genuinity\_calculator(x['rating'], x['positivity\_score']), axis=1)

data.head(20)

Output:



Practical-5

OBJECTIVE:

Simulation of Interpretability Machine Learning Algorithm

Tools Used:

* Python
* Vs code editor
* Jupyter Notebook
* Scikit-learn

Description:

In this practical we try to find the interpretability of logistic regression model on liver disease dataset by understanding how the predicted output label is depending on the input attributes that are used to decide whether the patient is having the liver disease or not.

Interpretable machine learning is the extraction of relevant knowledge from a machine-learning model concerning relationships either contained in data or learned by the model.

A common quote on model interpretability is that with an increase in model complexity, model interpretability goes down at least as fast. Feature importance is a basic (and often free) approach to interpreting your model. Even for black-box models such as deep learning, techniques exist to improve interpretability. Finally, the LIME framework will be discussed, which serves as a toolbox for model analysis.

Some popular ML interpretability Techniques

PDP: partial dependence plots

LIME: Local Interpretable Model-Agnostic

SHAP: Shapley additive exPlanations

CAM: Class Activation Mapping

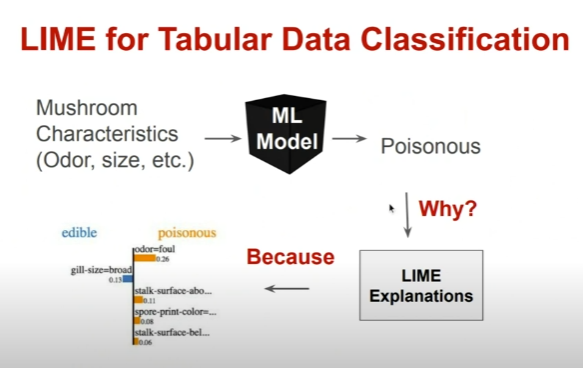
LIME

Local – explanations are locally faithful instead of globally.

Interpretable – Humans are limited by an amount of information that can be processed and understood

Model-Agnostivc – Any machine learning algorithm can be used as predictive model. Works with text, image and tabular data.

Explanations – Artifacts that provide an understanding between input to a ML model and the model’s prediction.



Code:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

%matplotlib inline

#load the data set using pandas

liver\_dataset = pd.read\_csv('indian\_liver\_patient.csv')

liver\_dataset.head(10)

liver\_dataset['Gender'] = liver\_dataset['Gender'].map({'Male': 1, 'Female': 2})

liver\_dataset['Albumin\_and\_Globulin\_Ratio'].fillna(value=liver\_dataset['Albumin\_and\_Globulin\_Ratio'].mean(),inplace=True)

X = liver\_dataset.drop(columns='Dataset', axis=1)

Y = liver\_dataset['Dataset']

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.4, random\_state=101)

from sklearn.linear\_model import LogisticRegression

model1 = LogisticRegression()

model1.fit(X\_train, Y\_train)

# accuracy on training data

from sklearn.metrics import accuracy\_score

X\_train\_prediction = model1.predict(X\_train)

training\_data\_accuracy = accuracy\_score(X\_train\_prediction, Y\_train)

print('Accuracy on Training data using Logistic Regression : ', training\_data\_accuracy\*100)

# accuracy on test data

X\_test\_prediction = model1.predict(X\_test)

test\_data\_accuracy = accuracy\_score(X\_test\_prediction, Y\_test)

print('Accuracy on Test data using Logistice Regression : ', test\_data\_accuracy\*100)

# Values from a Liver Scan

input\_data = (62,2,1.7,140,268,0,0,160,0,3.6)

# change the input data to a numpy array

input\_data\_as\_numpy\_array= np.asarray(input\_data)

# reshape the numpy array as we are predicting for only on instance

input\_data\_reshaped = input\_data\_as\_numpy\_array.reshape(1,-1)

#predicting data using Random Forest Classifier

prediction = model1.predict(input\_data\_reshaped)

if (prediction[0]== 2):

  print('According to the given details person does not have a Liver Disease')

else:

  print('According to the given details person has Liver Disease')

model1.coef\_

Output:

Text

Description automatically generated