

UNIVERSIDAD NACIONAL DEL ALTIPLANO
FACULTAD DE INGENIERÍA MECÁNICA ELÉCTRICA, ELECTRÓNICA Y SISTEMAS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS



Producto II Unidad

PREDICCIÓN DE DOLENCIAS A PARTIR DEL ESTADO DE SALUD ACTUAL DE UNA PERSONA



Nombres y Apellidos: Kevin Arnold Jallurani Ruelas
Título: Predicción De Dolencias A Partir Del Estado De Salud Actual De Una Persona
Docente: Ing. Donia Alizandra Ruelas Acero
Correo electrónico: kjalluranir@est.unap.edu.pe
Teléfono: 930913768
Fecha: 21, julio del 2023

Predicción De Dolencias A Partir Del Estado De Salud Actual De Una Persona

1. Descripción del Problema

En la actualidad, el diagnóstico temprano de enfermedades es fundamental para garantizar un tratamiento efectivo y mejorar la calidad de vida de las personas. Sin embargo, identificar las dolencias que experimenta una persona puede ser un desafío, ya que los síntomas pueden ser vagos o variados, y los pacientes pueden tener dificultades para expresarlos con precisión.

Este problema se aborda mediante el desarrollo de un sistema basado en Procesamiento del Lenguaje Natural (NLP) que tiene como objetivo predecir las dolencias de una persona en función de cómo se siente actualmente. El sistema utilizará algoritmos de aprendizaje automático y técnicas avanzadas de NLP para analizar y comprender la información proporcionada por el paciente sobre su estado de salud y síntomas.

El enfoque del sistema se basa en la recopilación de datos relevantes de los pacientes, como la descripción de sus síntomas, nivel de malestar, duración de los síntomas y otros factores relevantes. Estos datos se procesarán y analizarán para extraer patrones y relaciones que puedan indicar la posible dolencia o enfermedad que está experimentando el paciente.

Para lograr una mayor precisión en las predicciones, el sistema se entrenará con una amplia base de datos de casos médicos previos y diagnósticos confirmados. De esta manera, el modelo podrá aprender de ejemplos previos y desarrollar la capacidad de generalizar patrones para realizar nuevas predicciones.

Una vez desarrollado y probado el sistema, se espera que sea una herramienta útil para profesionales de la salud y médicos en particular. Al utilizar esta tecnología, los médicos podrán complementar su juicio clínico con información adicional y mejorar la toma de decisiones en el diagnóstico y tratamiento de los pacientes.

No obstante, es importante mencionar que el sistema de predicción de dolencias a partir del estado de salud actual de una persona mediante NLP no sustituirá la opinión de un profesional médico. Su objetivo es ser una herramienta de apoyo, proporcionando información valiosa para una evaluación más completa y rápida, lo que puede resultar en un tratamiento temprano más efectivo y, en última instancia, mejores resultados para los pacientes. La privacidad y seguridad de los datos de los pacientes serán primordiales en el diseño y funcionamiento del sistema, garantizando el cumplimiento de todas las regulaciones y estándares de protección de datos.

2. Descripción de los Datos

El dataset utilizado en este proyecto consta de dos columnas: "frase" e "indicación". Fue creado mediante la recopilación de información de consultas en línea donde las personas describen sus síntomas y malestares con el propósito de obtener una posible indicación médica. Cada fila representa una consulta distinta, con la "frase" conteniendo la descripción de los síntomas y el "indicación" proporcionando la posible dolencia o malestar sugerido por la consulta. Realizar una breve descripción de los datos utilizados.

El dataset es diverso y abarca una amplia variedad de síntomas y malestares reportados por diferentes individuos. Estas consultas y sus correspondientes indicaciones médicas se han obtenido de fuentes confiables en línea, como sitios web médicos, foros de salud y otras plataformas donde los usuarios buscan información y orientación sobre sus síntomas. Es importante tener en cuenta que, debido a la naturaleza de los datos recopilados en línea, el dataset podría contener errores, información incompleta o sesgos en la redacción. Por lo tanto, es fundamental realizar una limpieza y preprocesamiento adecuado de los datos antes de utilizarlos para entrenar el modelo de NLP.

Para mejorar la calidad y confiabilidad del dataset, se aplicarán técnicas de procesamiento de lenguaje natural para eliminar información sensible o identificable y garantizar la privacidad de los usuarios involucrados en las consultas. Asimismo, se considerará la posibilidad de incluir etiquetas adicionales o categorías de dolencias para mejorar la precisión del modelo.

El dataset será dividido en conjuntos de entrenamiento, validación y prueba para desarrollar y evaluar el modelo NLP. El objetivo final es crear un sistema de predicción de dolencias basado en NLP que pueda ayudar a los profesionales de la salud a obtener información relevante a partir de las descripciones de los síntomas proporcionadas por los pacientes, facilitando así el diagnóstico médico temprano y un tratamiento más efectivo.

3. Metodología

Google Colab: Google Colab (Colaboratory) es una plataforma en línea basada en Jupyter Notebook que permite ejecutar código Python en la nube. Es especialmente útil para trabajar en proyectos de ciencia de datos y aprendizaje automático, ya que proporciona acceso gratuito a recursos de computación como GPU y TPU. En el

contexto del código proporcionado, se utilizó Google Colab para ejecutar el código en un entorno de cuaderno colaborativo en la nube.

NLTK (Natural Language Toolkit): NLTK es una librería de Python ampliamente utilizada en el procesamiento del lenguaje natural. Proporciona una colección de herramientas y recursos para tareas como tokenización, lematización, etiquetado gramatical, análisis sintáctico, etc. En el código dado, la librería NLTK se utilizó para realizar el procesamiento de texto, específicamente con el stemmer SnowballStemmer para reducir las palabras a su raíz, lo que ayuda a normalizar el texto para el análisis posterior.

Coseno de Similitud: `cosine_similarity` es una función en la librería `scikit-learn` que mide la similitud coseno entre dos matrices o vectores. Hace la función de calcular el coseno del ángulo entre dos vectores en un espacio vectorial. Si el coseno de similitud es 1, los dos vectores tienen la misma dirección y son completamente similares; si es 0, los vectores son ortogonales y no tienen similitud; si es -1, tienen direcciones opuestas y son completamente diferentes.

Pandas: Pandas es una librería popular para el análisis y manipulación de datos en Python. Proporciona estructuras de datos como `DataFrame`, que son especialmente útiles para trabajar con conjuntos de datos tabulares. En el código dado, Pandas podría haberse utilizado para cargar y manipular el conjunto de datos, así como para realizar operaciones de limpieza y preprocesamiento.

Scikit-learn: Scikit-learn es una biblioteca ampliamente utilizada para el aprendizaje automático en Python. Ofrece una variedad de algoritmos de aprendizaje automático y herramientas para realizar tareas como clasificación, regresión, agrupación, selección de características, etc. En el código proporcionado, se utilizó Scikit-learn para crear y

entrenar el modelo RandomForestClassifier, un algoritmo de clasificación basado en árboles de decisión.

4. Resultados

Se cargaron las librerías necesarias para que nos ayude en toda la ejecución del código:

```
import numpy as np

import seaborn as sns

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.feature_extraction.text import
CountVectorizer,_preprocess,TfidfVectorizer

from sklearn.metrics.pairwise import linear_kernel,cosine_similarity

from nltk.stem.snowball import SnowballStemmer

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import *

from sklearn.model_selection import train_test_split
```


En esta línea de código, se carga un archivo CSV que contiene datos médicos utilizando la librería pandas. Los datos se almacenan en un DataFrame llamado medical_data.csv


```
data = pd.read_csv(r'medical_data.csv',low_memory=False)
```

En esta línea de código, se eliminan las filas duplicadas del DataFrame data, se restablecen los índices.

```
data = data.drop_duplicates().reset_index().drop('index',axis = 1)
```

Con el siguiente código “data” se va a lograr visualizar todos los datos que contiene el archivo csv para posteriores utilidades y para entrenarlos y hacer predicciones.

 data



	Phrase	Prompt
0	When I remember her I feel down	Emotional pain
1	there is too much pain when i move my arm	Heart hurts
2	My son had his lip pierced and it is swollen a...	Infected wound
3	My muscles in my lower back are aching	Infected wound
4	i have muscle pain that my back\nI Have Muscle...	Foot ache
...
735	There is a feeling of emptiness creeping up to...	emotional pain
736	This weird pain under my chest has been growin...	internal pain
737	It is very hard to concentrate since the last ...	emotional pain
738	Walking to work seems difficult now and there ...	body feels weak
739	My head aches badly when I try to look at my l...	head ache

740 rows × 2 columns

En el código proporcionado, se definen dos variables:

`punctuation=['\"'?'',\']`: Esta variable contiene una expresión regular que representa una lista de caracteres de puntuación que se desean reemplazar en el texto. La expresión regular incluye los caracteres ", ', ?, , y ..

`abbr_dict`: Esta variable es un diccionario que contiene diversas abreviaturas y sus correspondientes expansiones completas. Estas abreviaturas serán utilizadas para reemplazar las versiones abreviadas en el texto durante el preprocesamiento.

```
punctuation=['\"'?'',\'] # I will replace all these punctuation with "
```

```
abbr_dict={  
    "what's":"what is",  
    "what're":"what are",  
    "where's":"where is",  
    "where're":"where are",  
    "i'm":"i am",  
    "we're":"we are",  
    "it's":"it is",  
    "that's":"that is",  
    "there's":"there is",  
    "there're":"there are",  
    "i've":"i have",  
    "who've":"who have",  
    "would've":"would have",  
    "not've":"not have",  
    "i'll":"i will",  
    "it'll":"it will",  
    "isn't":"is not",  
    "wasn't":"was not",  
    "aren't":"are not",  
    "weren't":"were not",  
    "can't":"can not",  
    "couldn't":"could not",  
    "don't":"do not",  
    "didn't":"did not",
```



```

"shouldn't":"should not",
"wouldn't":"would not",
"doesn't":"does not",
"haven't":"have not",
"hasn't":"has not",
"hadn't":"had not",
"won't":"will not",
punctuation:",
\s+:' ', # replace multi space with one single space
}

```

En la siguiente línea de código, La función `process_data` realiza las siguientes tareas de preprocesamiento en el DataFrame `data`: Convierte todas las cadenas de texto en minúsculas para uniformizar el formato. Asegura que todas las columnas sean tratadas como cadenas de texto. Reemplaza las abreviaturas presentes en el texto utilizando un diccionario de expansión. Aplica el stemming (reducción a su raíz) a las palabras en la columna 'Phrase', creando una nueva columna llamada 'stemmed_phrase'. Muestra las primeras 10 filas del DataFrame después del preprocesamiento.

```

def process_data(data):
    # Convert to lower case

    data.Phrase=data.Phrase.str.lower()
    data.Prompt=data.Prompt.str.lower()

    # convert to string
    data.Phrase=data.Phrase.astype(str)

```

```

data.Prompt=data.Prompt.astype(str)

# replace abbreviations

data.replace(abbr_dict,regex=True,inplace=True)


#apply stemming

stemmer = SnowballStemmer("english")

data['stemmed_phrase'] = data['Phrase'].apply(lambda x : ''.join([stemmer.stem(y)
for y in x.split()])))

display(data.head(10))

return data

```

La línea de código **data = process_data(data)** aplica el preprocesamiento a los datos almacenados en el DataFrame data, incluyendo la conversión a minúsculas, la expansión de abreviaturas y el stemming de las frases en la columna 'Phrase'. Después de esta línea, el DataFrame data contiene los datos preprocesados.

data = process_data(data)

	Phrase	Prompt	stemmed_phrase
0	when i remember her i feel down	emotional pain	when i rememb her i feel down
1	there is too much pain when i move my arm	heart hurts	there is too much pain when i move my arm
2	my son had his lip pierced and it is swollen a...	infected wound	my son had his lip pierc and it is swollen and...
3	my muscles in my lower back are aching	infected wound	my muscl in my lower back are ach
4	i have muscle pain that my back i have muscle ...	foot ache	i have muscl pain that my back i have muscl pa...
5	i have muscle pain in my left leg	shoulder pain	i have muscl pain in my left leg
6	i have cut my finger because of playing footba...	injury from sports	i have cut my finger becaus of play footbal an...
7	i have acne in my face and other problems in m...	skin issue	i have acn in my face and other problem in my ...
8	i have a strange rash on my arm	foot ache	i have a strang rash on my arm
9	i have a sharp pain in my lower stomach	stomach ache	i have a sharp pain in my lower stomach

d2 = data[['stemmed_phrase', 'Prompt']]: Se seleccionan dos columnas ('stemmed_phrase' y 'Prompt') del DataFrame data para formar un nuevo DataFrame llamado d2.

d2.to_csv('trial_data.csv'): El DataFrame d2 se guarda en un archivo CSV llamado 'trial_data.csv'. **ailments = data['Prompt'].unique():** Se obtiene una lista de valores únicos de la columna 'Prompt' del DataFrame data, que representa las diferentes dolencias o indicaciones presentes en los datos.

```
[ ] d2 = data[['stemmed_phrase', 'Prompt']]
```

```
[ ] d2.to_csv('trial_data.csv')
```

```
[ ] ailments = data['Prompt'].unique()
```

Se carga un archivo CSV en un DataFrame llamado data. Luego, se eliminan filas duplicadas del DataFrame data. A continuación, se define un diccionario de expansión de abreviaturas y una función para preprocesar los datos. Los datos se preprocesan y se guardan en el DataFrame data. Se crea un nuevo DataFrame d2 con dos columnas seleccionadas de data, que luego se guarda en un archivo CSV. Además, se obtiene una lista de las diferentes dolencias presentes en los datos y se crean dos diccionarios para contar las ocurrencias de las dolencias y asignarles un número único.

```
dict_ail = {}

# for a in ailments:
#     dict_ail[a] = 0

for k in data.index:
    name = data['Prompt'][k]
    dict_ail[name] = dict_ail.get(name, 0) + 1
```

```

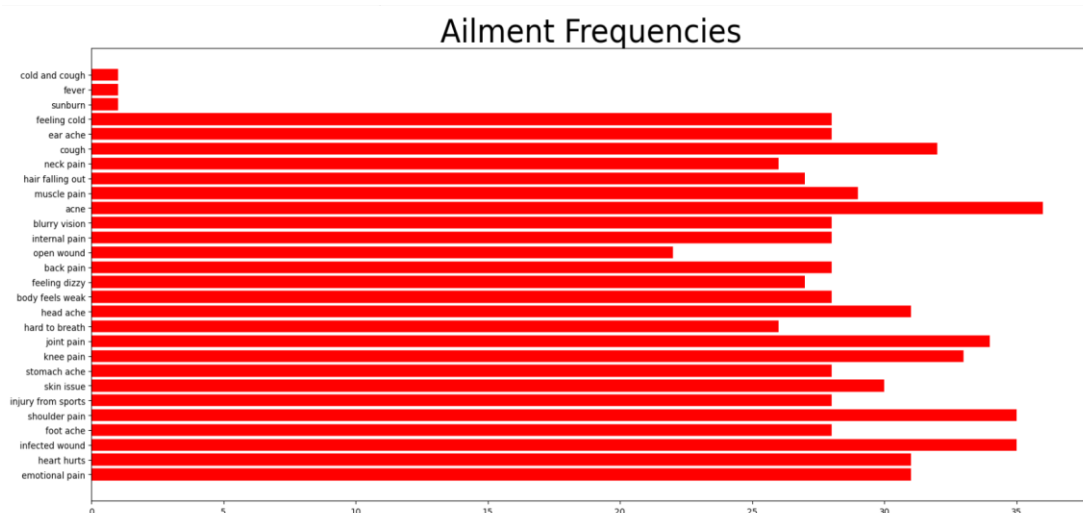
ailment_dict = {}

for i,k in enumerate(dict_ail.keys()):

    ailment_dict[i] = k

```

Se crea una figura con un tamaño de 18x8 pulgadas. Se le asigna un título "Ailment Frequencies" con un tamaño de fuente de 35. Se dibuja un gráfico de barras horizontales con colores rojos. El eje y se establece con valores numéricos que representan las posiciones de las dolencias en la lista. Las anchuras de las barras se definen utilizando los valores del diccionario dict_ail. Los rótulos del eje y se toman de las claves del diccionario dict_ail. Finalmente, se ajusta el diseño del gráfico para asegurarse de que los elementos estén bien distribuidos y visibles.



Se crean dos transformadores de características para el procesamiento de texto: `Cv` utiliza `CountVectorizer` y `Tfidf` utiliza `TfidfVectorizer`. Se aplican a los datos procesados para obtener las matrices de características `transformed_count` y `transformed_idf`. Luego, se aplica `Tfidf` al texto de ejemplo "I am experiencing pain in

the leg from the past two days" para obtener la matriz de características trial con la ponderación TF-IDF.

```
[ ] Cv = CountVectorizer(stop_words='english', ngram_range = (1,3), max_df=0.7)
    transformed_count = Cv.fit_transform(data['stemmed_phrase'])

[ ] Tfidf = TfidfVectorizer(stop_words = 'english', ngram_range= (1,3), max_df= 0.7)
    transformed_idf = Tfidf.fit_transform(data['stemmed_phrase'])

[ ] input_text = ['I am experiencing pain in the leg from the past two days']
    trial = Tfidf.transform(input_text)

[ ] trial

<1x4970 sparse matrix of type '<class 'numpy.float64'>'
    with 4 stored elements in Compressed Sparse Row format>
```

ailment_dict es un diccionario que asigna un número único a cada dolencia presente en los datos. La clave de cada entrada es el número único asignado y el valor es el nombre de la dolencia correspondiente.

```
ailment_dict
{0: 'emotional pain',
 1: 'heart hurts',
 2: 'infected wound',
 3: 'foot ache',
 4: 'shoulder pain',
 5: 'injury from sports',
 6: 'skin issue',
 7: 'stomach ache',
 8: 'knee pain',
 9: 'joint pain',
10: 'hard to breath',
11: 'head ache',
12: 'body feels weak',
13: 'feeling dizzy',
14: 'back pain',
15: 'open wound',
16: 'internal pain',
17: 'blurry vision',
18: 'acne',
19: 'muscle pain',
20: 'hair falling out',
21: 'neck pain',
22: 'cough',
23: 'ear ache',
24: 'feeling cold',
25: 'sunburn',
26: 'fever',
27: 'cold and cough'}
```

Por siguiente, La consulta se procesa primero y se convierte a minúsculas. La variable query contiene la consulta original "From past few weeks feeling sad".

```
# the query is first processed and made into lower case
```

```
query = "From past few weeks feeling sad"
```

La función process_query(query) toma una consulta como entrada y realiza las siguientes acciones:

- Convierte la consulta a minúsculas.
- Reemplaza las abreviaturas presentes en la consulta con sus formas completas utilizando el diccionario abbr_dict.
- Realiza el proceso de stemming a través del stemmer SnowballStemmer de NLTK para reducir las palabras a su forma base.
- Retorna la consulta preprocesada y lista para ser utilizada en el modelo.

```
def process_query(query):
```

```
    # Change to lower
```

```
    query = query.lower()
```

```
    # Removed abbreviations
```

```
    res = ""
```

```
    # print(query.split())
```

```
    for k in query.split():
```

```
        if k in abbr_dict:
```

```
            print(abbr_dict[k])
```

```
            res+=' ' + abbr_dict[k]
```

```
        else:
```

```
            res+=' ' + k
```

```

stemmer = SnowballStemmer('english')

res = ''.join([stemmer.stem(y) for y in res.split()])

return res

```

El código imprime "Example query:" y luego imprime la consulta original y la consulta preprocesada utilizando la función `process_query(query)`.

Por ejemplo:

Si la consulta original es "From past few weeks feeling sad", la consulta preprocesada sería "from past few week feel sad". Esto se logra convirtiendo la consulta a minúsculas, eliminando abreviaturas y aplicando el proceso de stemming para reducir las palabras a su forma base.

```

[ ] print("Example query: ")
    print("Final query:", process_query(query))
    processed = process_query(query)

```

```

Example query:
Final query: from past few week feel sad

```

El código calcula la similitud coseno entre la consulta preprocesada y las frases del conjunto de datos, y luego selecciona las tres frases más similares a la consulta.

```

query = [processed]

res = TfIdf.transform(query)

sim = cosine_similarity(res, transformed_idf)

res = list(np.argsort(sim))[0]

res = res[::-1][:3]

```

El código imprime las tres dolencias más similares a la consulta preprocesada. Utiliza la matriz de similitud coseno para identificar las frases con mayor similitud y luego muestra las dolencias correspondientes a esas frases.

```

▶ for k in res:
    print(data.loc[k]['Prompt'])

```

↗ emotional pain
 emotional pain
 skin issue

La función `get_prediction(query)` toma una consulta, la procesa y utiliza la similitud coseno para encontrar las frases más similares. Luego devuelve la dolencia más probable para la consulta.

```

def get_prediction(query):

    print("Query is :",query)

    processed = process_query(query)

    query = [processed]

    print("Processed :",query)

    res = Tfidf.transform(query)

    sim = cosine_similarity(res,transformed_idf)

    res = list(np.argsort(sim))[0]

    res = res[::-1][:20]

    print(sim[0][res[0]],sim[0][res[1]])

    ailment =[]

    # let's find most similar sentences and then see

    # use levenshtein distance after you have got the result

    for k in res[1]:

```



```
ailment.append(data.loc[k]['Prompt'])

print("Results :")

return ailment
```

```
for q in data['stemmed_phrase'][500:]:

    print(get_prediction(q))
```

Itera a través de las frases preprocesadas desde la posición 500 hasta el final en el DataFrame data. Para cada frase, llama a la función `get_prediction(q)` para obtener la dolencia más probable asociada a esa frase. Imprime la dolencia más probable para cada frase.

Se crea un modelo de clasificación `RandomForestClassifier` con los siguientes hiperparámetros:

- `n_estimators=100`: Indica el número de árboles de decisión que se utilizarán en el ensamblado del modelo.
- `min_samples_leaf=2`: Representa el número mínimo de muestras requeridas para que un nodo sea considerado una hoja en un árbol.
- `bootstrap=True`: Indica si se realiza muestreo con reemplazo para construir los árboles individuales en el bosque aleatorio.

```
model =

RandomForestClassifier(n_estimators=100,min_samples_leaf=2,bootstrap=True)
```

Se crea un objeto `TfidfVectorizer` que convierte las frases preprocesadas en una matriz de características usando la técnica TF-IDF. Luego, se obtiene la representación vectorial de las frases en la matriz X.

```
Tfidf = TfidfVectorizer(stop_words = 'english', ngram_range= (1,3),max_df= 0.7)
X = Tfidf.fit_transform(data['stemmed_phrase']).toarray()
```

Se crea un diccionario llamado `ailment` que asigna un índice numérico a cada clase de dolencia en el diccionario `ailment_dict`. Luego, se crea una nueva serie `Y` que mapea las clases de dolencias en el DataFrame `data['Prompt']` a sus respectivos índices numéricos utilizando el diccionario `ailment`.

```
# ailment_dict
ailment = {}
for i,j in ailment_dict.items():
    ailment[j] = i
print(ailment)
Y = data['Prompt'].map(ailment)
```

Se divide la matriz de características `X` y la serie de etiquetas `Y` en conjuntos de entrenamiento (`X_train` y `Y_train`) y conjuntos de prueba (`X_test` y `Y_test`). Se utiliza una proporción del 80% para el entrenamiento y el 20% para las pruebas. El parámetro `random_state` se utiliza para asegurar la reproducibilidad de los resultados y el parámetro `shuffle` se establece en `True` para mezclar aleatoriamente los datos antes de dividirlos.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size = 0.8,
random_state = 43, shuffle = True)
```

Se entrena el modelo `RandomForestClassifier` con los datos de entrenamiento `X_train` y las etiquetas de entrenamiento `Y_train`. El modelo aprenderá a realizar predicciones

basadas en las características proporcionadas en `X_train` y las correspondientes etiquetas en `Y_train`.

```
model.fit(X_train,Y_train)
```

Se hacen predicciones para el conjunto de prueba `X_test` utilizando el modelo entrenado, y los resultados se almacenan en la variable `y_preds`.

```
y_preds = model.predict(X_test)
```

Se cuenta el número de predicciones correctas e incorrectas comparando las predicciones del modelo con las etiquetas reales del conjunto de prueba.

```
correct,incorrect =0,0
for k,i in zip(y_preds,Y_test):
    if(k==i):
        correct+=1
    else:
        incorrect+=1
```

Se obtuvo el siguiente resultado:

```
[ ] correct
```

```
102
```

```
[ ] incorrect
```

```
46
```

Se evalúa la precisión del modelo RandomForestClassifier con diferentes valores de `n_estimators` y se grafican los resultados.

```
score=[]

for est in range(10,50):

    model = RandomForestClassifier(n_estimators=est,min_samples_leaf=2)

    model.fit(X_train,Y_train)

    s = model.score(X_test,Y_test)

    score.append(s)

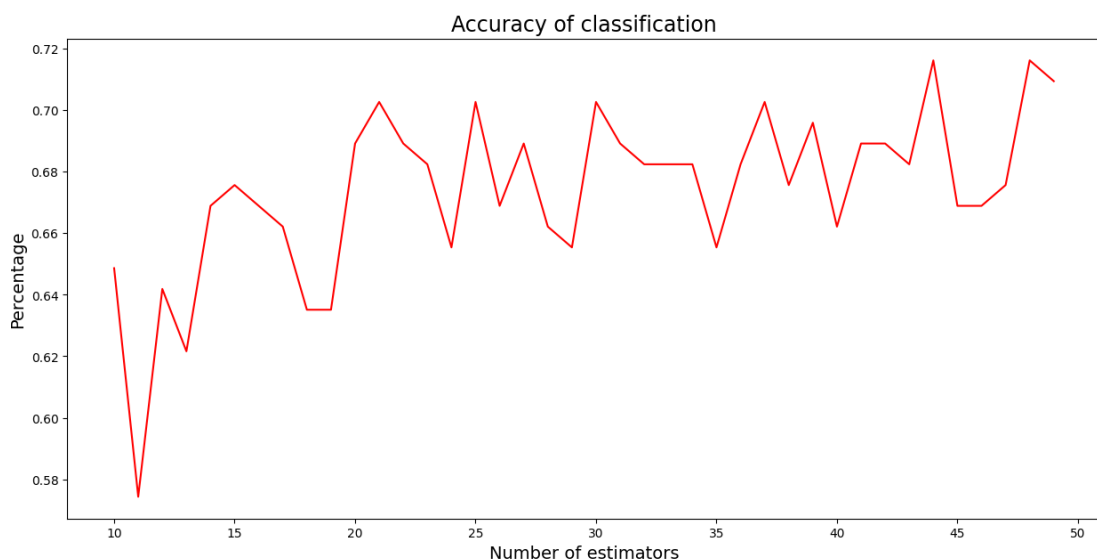
plt.figure(figsize= (15,7))

plt.title("Accuracy of classification",fontsize=17)

plt.xlabel("Number of estimators",fontsize = 14)

plt.ylabel("Percentage",fontsize = 14)

plt.plot([i for i in range(10,50)],score,color= 'red')
```



Y por último, este código implementa un sistema de recomendación de enfermedades en base a una consulta del usuario utilizando el modelo RandomForestClassifier previamente entrenado. El sistema procesa la consulta del usuario, realiza la

vectorización mediante TfIdf, y luego utiliza el modelo para predecir las tres enfermedades más probables que corresponden a la consulta. Finalmente, muestra las enfermedades predichas al usuario. El ciclo se repite hasta que el usuario escribe "salir" para terminar la interacción.

Para realizar las pruebas vamos a tomar unas palabras de la data set para ver que malestares o dolencias nos predice.



Obtenemos las siguientes predicciones:

```
Ingresas tu consulta (o escribe 'salir' para terminar): My back hurts me and i can't bend it or walk
Consulta procesada: my back hurt me and i can not bend it or walk
Clases predichas: back pain, knee pain, foot ache
```

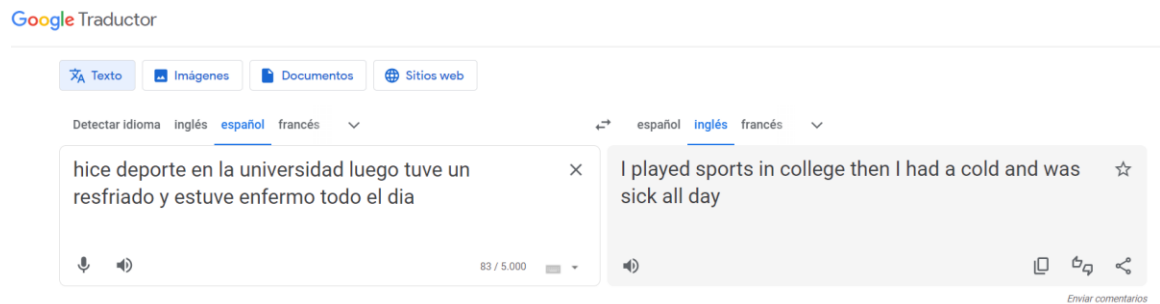
Donde se determina que:

Ingresas tu consulta (o escribe 'salir' para terminar): Me duele la espalda y no puedo doblarla ni caminar

Consulta procesada: me duele la espalda y no puedo doblarla ni caminar

Clases predichas: dolor de espalda, dolor de rodilla, dolor de pie

Otra prueba con palabras que no se encuentran en el data set y son elaborados para un testeo del modelo.



Obtenemos las siguientes predicciones:

```
Ingresa tu consulta (o escribe 'salir' para terminar): I played sports in college then I had a cold and was sick all day
Consulta procesada: i play sport in colleg then i had a cold and was sick all day
Clases predichas: injury from sports, feeling cold, head ache
Ingresa tu consulta (o escribe 'salir' para terminar):
```

Donde se determina que:

Ingresa tu consulta (o escribe 'salir' para terminar): Practiqué deportes en la universidad, luego tuve un resfriado y estuve enfermo todo el día.

Consulta procesada: hago deporte en la universidad luego me resfrí y estuve todo el día enferma

Clases predichas: lesión por deporte, sensación de frío, dolor de cabeza

5. Conclusión

Se llegó a las conclusiones de que con la implementación del código es que se ha desarrollado un sistema de recomendación de enfermedades basado en el estado de salud actual de una persona, expresado en forma de texto. El sistema utiliza técnicas de procesamiento del lenguaje natural y aprendizaje automático para procesar la consulta del usuario, identificar las enfermedades más probables relacionadas con esa consulta y ofrecer al usuario una lista de las tres enfermedades más relevantes.

El modelo de clasificación Random Forest ha sido entrenado con datos previos de enfermedades y síntomas para aprender patrones y relaciones entre el texto y las enfermedades. Luego, el modelo se utiliza para realizar predicciones sobre nuevas consultas de los usuarios.

El sistema proporciona una forma interactiva para que los usuarios ingresen sus síntomas o describan su estado de salud actual en palabras simples. A partir de esta información textual, el sistema recomienda posibles enfermedades que podrían estar relacionadas con los síntomas o la descripción proporcionada.

Es importante mencionar que el rendimiento y la precisión del sistema de recomendación dependen en gran medida de la calidad del conjunto de datos utilizado para entrenar el modelo y la efectividad del procesamiento del lenguaje natural aplicado. En aplicaciones del mundo real, se requeriría un conjunto de datos médicos representativo y diversificado, así como técnicas de procesamiento de texto cuidadosamente seleccionadas para obtener resultados confiables y precisos.

6. Anexos

- Enlace a código fuente:

https://github.com/KevinJalluranix/NLP_Prediccion_estado_de_salud.git

- Enlace del colab:

https://colab.research.google.com/drive/1QeWpwZRB7mFIEx9U_bCZQTntEcQgQz2s?usp=sharing