# Lab 7 - Classifier Design & Cleaning Text Data

**Classifier design** ia the process of building a model that can classify data into different classes or categories. The input data can take various forms such as text, images, audio, or numerical data. It involves various steps, including selecting a suitable machine learning algorithm, defining features or input variables, collecting and preparing training data, training the model on the training data, evaluating the model's performance on test data, and tuning the model to improve its performance.

The ultimate goal of classifier design is to develop a model that can accurately predict the class labels of new, unseen data based on its characteristics. Classifier design has wide range of applications in fields such as image recognition, natural language processing, and fraud detection.

# Heart Disease UCI

subset of the 74 attributes in the original: https://archive.ics.uci.edu/ml/datasets/heart+disease (https://archive.ics.uci.edu/ml/datasets/heart+disease)

Attribute Information:

- age
- sex
- chest pain type (4 values)
- resting blood pressure
- serum cholestoral in mg/dl
- fasting blood sugar > 120 mg/dl
- resting electrocardiographic results (values 0,1,2)
- maximum heart rate achieved
- exercise induced angina
- oldpeak = ST depression induced by exercise relative to rest
- the slope of the peak exercise ST segment
- number of major vessels (0-3) colored by flourosopy
- thal: 3 = normal; 6 = fixed defect; 7 = reversable defect

We will learn how to use classifiers to train and predict labels for this dataset.

In [ ]:
```
import pandas as pd
import numpy as np

#Load the data using pandas read_csv function.
orig_data = pd.read_csv("heart.csv")
orig_data.head()
```

In [ ]:
```
#get the features in X and store the target column in y .
X = orig_data.iloc[:, :-1]
#extract the target column.
y = orig_data["target"]
```

Split the dataset into train and test

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

You may wish to train some scaling or preprocessing transformers.

- Conisder the standard scaler

The `StandardScalar` is a pre-processing transformer in scikit-learn that standardizes a dataset along each feature dimension by removing the mean and scaling to unit variance.This ensures that each feature contributes equally to the model's performance, aand it helps to improve the numerical stability of the machine learning model.

In [ ]:
```
from sklearn.preprocessing import StandardScaler
#we can use a transformer to scale our data
transformer = StandardScaler()
X_train = transformer.fit_transform(X_train)
X_test = transformer.transform(X_test)
```

Let's build a number of individual classifiers to train and use on the data.

Try to build 3 standard classifiers **LogisticRegression,DecisionTreeClassifier,RandomForestClassifier**

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier

clf1 = LogisticRegression()
clf2 = DecisionTreeClassifier(max_depth = 4,random_state = 0)
clf3 = RandomForestClassifier(n_estimators=100,random_state=0)
```

In [ ]:
```
clf1.fit(X_train, y_train)
clf2.fit(X_train, y_train)
clf3.fit(X_train, y_train)
y_pred1=clf1.predict(X_test)
y_pred2=clf2.predict(X_test) #Clf 2 is the decision tree model
y_pred3=clf3.predict(X_test)
```

## Lets Expand a bit more on the decision tree classifier

In [ ]:
```
target_names = ['No_heart_problem', 'heart_disease']
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
cm = confusion_matrix(y_test, y_pred2)

#we can plot it
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clf2, X_test, y_test)
```

In [ ]:
```
#Classification Report

print(classification_report(y_test, y_pred2, target_names=target_names))
```

In [ ]:
```
accuracy_score(y_test,y_pred2)
```

In [ ]:
```
accuracy_score(y_test,y_pred1)
```

Try various accuracy scores for other classifiers

# Cleaning Text Data - Important for coursework

## Data loading and initial exploration

We begin by loading a dataset that you can obtain from Kaggle. The database can be found at https://www.kaggle.com/andrewmvd/trip-advisor-hotel-reviews (https://www.kaggle.com/andrewmvd/trip-advisor-hotel-reviews) and contains reviews of hotels crawled from Tripadvisor. You can explore what makes a great hotel and maybe even use any model you develop to plan your next trip :-)

As you will see the dataset has two fields or attributes, the review *text* and the review *rating*. We could use the rating as a classification or regression variable.

The `encoding = utf-8` parameter in the `read_csv()` function specifies encoding of the CSV file. UTF-8 is a widely used character encoding that can handle most of hte characters in any language.

In [ ]:
```
#Load the data using pandas read_csv function.
df= pd.read_csv("tripadvisor_hotel_reviews.csv", encoding='utf-8')

#We set to first explore a small sample of 100 reviews. At the end you could include more reviews.
df=df.sample(100)
print(df.head())
print(df.shape)
```

In [ ]:
```
df['Rating'].value_counts()
```

We can reclassify the target class to obtain less classes, which may be easier to classify. We can classify 1 and 2 as low, 3 as medium and 4 or 5 as high. We will still have unbalanced classes as we can see by the counts. We can also keep a copy of the review attribute so that as we change it, we can keep the original values.

In [ ]:
```
# We will copy the review in another column so that the original text is also there for comparison

df["text"] = df["Review"].astype(str)

# Data has 5 classes, let's convert them to 3

def classes_def(x):
    if x ==  1 or x==2:
        return "Low"
    elif x == 3:
        return "Med"
    elif x == 4 or x == 5:
        return "High"
df['rate']=df['Rating'].apply(lambda x:classes_def(x))
target=df['rate']

df['rate'].value_counts()
```

## Data preprocessing

Now we start by taking out things that may not be helpful to the analysis. We could start by taking out for example urls. This may not be so prominent in this data, but they may be if we are analysing Tweets as we may do later. We can use the Regula Expression `re` module for this, and we can do matching of regular expression, to find anything to remove. For example `re.sub(pattern, repl, string, count=0,` `flags=0)` returns the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl. If the pattern isn’t found, string is returned unchanged.

In [ ]:
```
import re#regular expressions: https://book.pythontips.com/en/latest/lambdas.html

#Removes the urls on the column 'text'
df['text'] = df['text'].apply(lambda x:re.sub(r'\s*https?://t\.co/[a-zA-Z0-9]+\s*', ' ', x))
print(df['text'])
```

In [ ]:
```
# We can also remove any numbers, as they may not be useful to the analysis, using a similar approach.
df['text'] = df['text'].apply(lambda x:re.sub(r'[0-9]+', '', x))
print(df['text'])
```

We now break down the text into a token list, which we can then use to apply various operations including stemming, etc. We can use the `word_tokenize` function from `nltk.tokenize` to extract the words. Note that at this point the extracted tokens include puntuaction marks, etc.

In [ ]:
```
# Importing necessary library
import nltk.corpus #natural language toolkit
# import nltk.corpus
nltk.download('stopwords')
nltk.download('punkt')
import os
from nltk.tokenize import word_tokenize

# Passing the string text into word tokenize for breaking the sentences
Token_list=[]
for tweet in df['text']:
    Token_list.append(word_tokenize(tweet))

#Let us look at the content of the first review as a set of tokens, and the rating.
print(Token_list[0])
print(df.iloc[0:1])
```

In [ ]:
```
#Remove the puntuation and other non alphanumeric tokens
for i in range (0,len(Token_list)):
    Token_list[i] =[word for word in Token_list[i] if word.isalnum() ]

#Again we look at first review to see the effect.
print(Token_list[0])
```

In [ ]:
```

stopwords = nltk.corpus.stopwords.words('english')
print(stopwords)# pronouns and prepositions mainly
# importing stopwors from nltk library
from nltk import word_tokenize
from nltk.corpus import stopwords

a = set(stopwords.words('english'))

for i in range (0,len(Token_list)):
    Token_list[i] =[word for word in Token_list[i] if word not in a ]
print(Token_list[0])

#print(stopwords)
```

So far we have looked at words, which are also known as *1-grams*. Sometimes, we want to group consecutive words together to capture more complex concepts. One way to do this will be to use n-grams, which could be 2-grams for 2 consecutive words, 3-grams for 3 consecutive words, etc. We look at those here by using the `ngrams` function and we print 2-grams and 3-grams for the first review to understand what they may look like. In some applications it may be that 2-grams or 3-grams may be more informative.

In [ ]:
```
from nltk.util import ngrams

def words_to_ngrams(words, n, sep=" "):
    return [sep.join(words[i:i+n]) for i in range(len(words)-n+1)]

print("Bigrams: ", words_to_ngrams(Token_list[0],2))
print("\nTrigrams: ",words_to_ngrams(Token_list[0],3))
```

We are going to focus on the 1-gram or words for the rest of this exercise. Let us calculate the most frequent tokens (in this case words) using the `FreqDist` function which gives us the word or token and its frequency.

In [ ]:
```
# finding the frequency distinct in the tokens
# Importing FreqDist library from nltk and passing token into FreqDist
from nltk.probability import FreqDist
fdist = FreqDist()
for i in range (0,len(Token_list)):
    for word in Token_list[i]:
        fdist[word.lower()]+=1
fdist
```

In [ ]:
```
#Concatenate the words together for each document or review
word_concat=[' '.join(word) for word in  Token_list]
```

## TF-IDF

We can however try to extract features using tf-idf . Let us do that by using the `TfidfVectorizer`

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tf=TfidfVectorizer()
text_tf= tf.fit_transform(word_concat)

print (text_tf.todense())
```

In [ ]:
```
X= text_tf
```

In [ ]:
```
y=df['rate']
from sklearn.preprocessing import LabelEncoder

#encode the labels from strings to values.
le = LabelEncoder()

y = le.fit_transform(y)
print(le.classes_)
y
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Try a Random Forest classifier and look at results.

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(criterion='entropy',max_depth= 2,class_weight='balanced')
clf.fit(X_train,y_train);
y_pred = clf.predict(X_test)
y_pred
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

#we can get the confusion matrix
cm = confusion_matrix(y_test, y_pred)
#we can plot it
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clf, X_test, y_test)

print('Results on the test set:')
print(classification_report(y_pred, y_test))
```

## Creating Wordclouds

A word cloud is a visual representation of textual data in which the size of each word indicates its frequency or importance in the text. Typically, the more frequently a word appears in the text, the larger and bolder it is in the word cloud.

It can be useful in quickly summarizing the key themes or topics in a large body of text, such as customer reviews or social media posts and also be used for visualizing the results of text mining or natural language processing techniques.

In [ ]:
```
words = ''

# iterate through the text attribute in the dataframe splitting the text into the component tokens
for val in df['text'] :
    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    words += " ".join(tokens)+" "
```

In [ ]:
```
# importing all necessery modules
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = a,#We don't want to see stopwords in the word clous
                min_font_size = 10,
                max_font_size=100,
                max_words=50).generate(words)


# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

Please try positive and negative wordcloud by yourself

In [ ]: