Medical Diagnosis using Support Vector Machines

1. Load a dataset from file
2. Split a dataset into training and testing subsets
3. Normalize Features(**Standard Scaler**)
4. Create a **support vector machine**
5. **Hyperparameter** Optimization
6. Make a medical diagnosis for a new patient
7. Evaluate the accuracy of the **SVM classifier (Precision,recall)**

Analyze Box Office Data with Seaborn and Python

1. Data Loading and Exploration
2. Visualizing the Target Distribution (**dist plot**)
3. Relationship between Film Revenue to Budget(**scatter plot**)
4. Do Official Homepages Impact Revenue? (**categorical plot**)
5. Distribution of Languages across Films (**boxplot – revenue(log) vs language**)
6. Frequent Words in Film Titles and Descriptions(**WordCloud**)
7. How do Film Descriptions Impact Revenue? (**Tfidf Vectorizer, Linear Regression**)

Perform Sentiment Analysis with scikit-learn

NLP

1. Introduction and Importing the Data
2. Transforming Documents into **Feature Vectors (Bag of words / Bag of N-grams model)**
3. **Word Relevancy** using Term Frequency-Inverse Document Frequency
4. Calculate **TF-IDF** of the Term 'Is'
5. Data Preparation
6. **Tokenization of Documents (Stemming, Removing Stopwords)**
7. Transform Text Data into **TF-IDF Vectors**
8. Document Classification Using **Logistic Regression**
9. Load Saved Model from Disk
10. Model Accuracy (**Accuracy score**)

Fake News Detection with Machine Learning

NLP

1. Understand the Problem Statement and business case
2. **Import libraries and datasets**
3. Perform Exploratory Data Analysis
4. Perform Data Cleaning (**Removing stopwords**)
5. Visualize the cleaned data (**countplot**- subject, **wordcloud**)
6. Prepare the data by **tokenizing and padding (tokenization to text_to_sequences)**
7. Build and train the model(**Sequential Model- use adam optimizer** , add **Bi-Directional RNN and LSTM**)
8. Assess trained model performance (**Accuracy score, Confusion matrix**)

Basic Image Classification with TensorFlow

NN

1. Import libraries and dataset(**Tensorflow as tf, mnist**)
2. The Dataset
3. One hot encoding (**Encoding labels**)
4. Preprocess image examples (**Unrolling N-dimensional Arrays to Vectors, Data Normalization**)
5. Create a neural network model (Sequential, **activation- relu,softmax, compiling it**)
6. Train the model to fit the dataset
7. Evaluate the model (~96)
8. Visualize the predictions

- Support Vector Machine Algorithm

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane.

- **Bag of Words** model

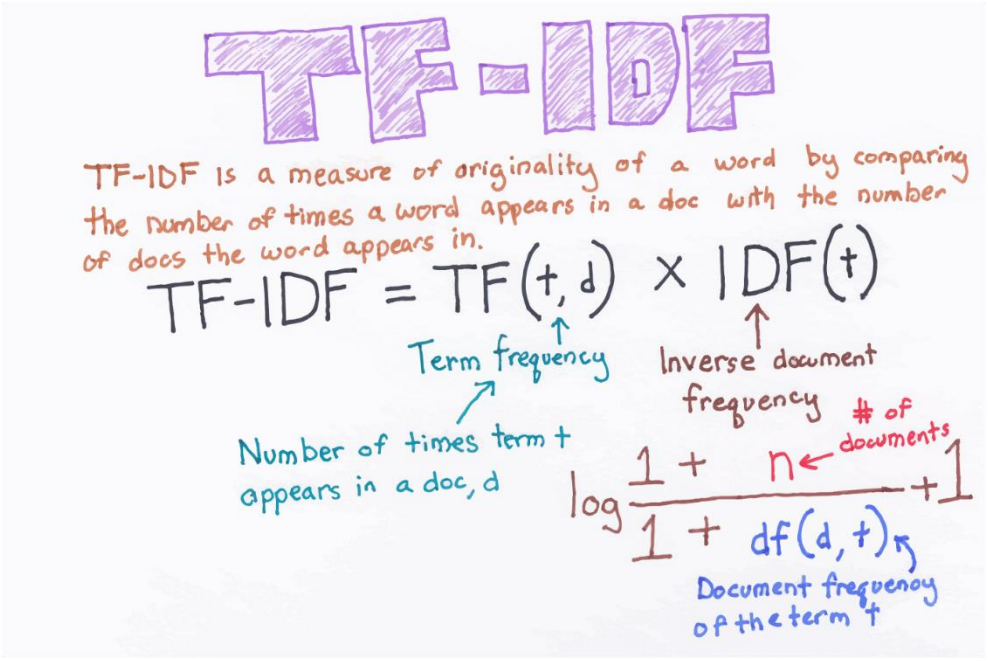
It's an algorithm that transforms the text into fixed-length vectors. This is possible by counting the number of times the word is present in a document. The word occurrences allow to compare different documents and evaluate their similarities for applications, such as search, document classification, and topic modeling.

The reason for its name, "Bag-Of-Words", is due to the fact that it represents the sentence as a bag of terms. It doesn't take into account the order and the structure of the words, but it only checks if the words appear in the document.

In the table, I show all the calculations to obtain the Bag-Of-Words approach:

	amazing	an	best	game	great	is	of	series	so	the	thrones	tv
0	1	1	0	1	0	1	1	1	0	0	1	1
1	0	0	1	1	0	1	1	1	0	1	1	1
2	0	0	0	1	1	1	1	0	1	0	1	0

TF(Term Frequency)-IDF(Inverse Document Frequency)



TF-IDF computes the weight of a word in a specific document, taking into account the overall distribution of words.

TF-IDF stands for Term Frequency Inverse Document Frequency of records. It can be defined as the calculation of how relevant a word in a series or corpus is to a text. The meaning increases proportionally to the number of times in the text a word appears but is compensated by the word frequency in the corpus (data-set).

TF is frequency counter for a term t in document d, where as DF is the count of **occurrences** of term t in the document set N. In other words, DF is the number of documents in which the word is present.

Tokenization in NLP

Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.

Natural Language Processing

['Natural', 'Language', 'Processing']

We can use this tokenized form to:

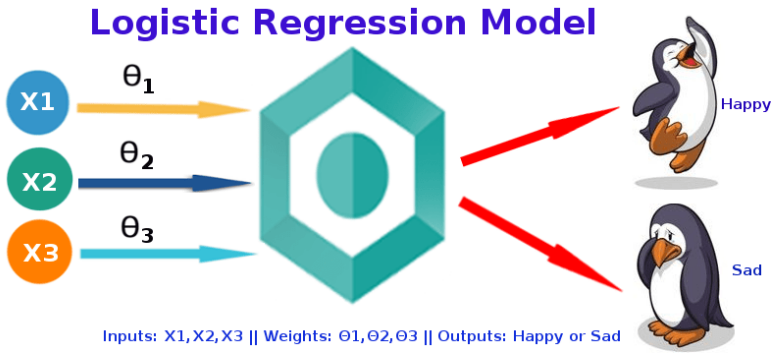
- Count the number of words in the text
- Count the frequency of the word, that is, the number of times a particular word is present

Tokenization using Python's split() function
Tokenization using Regular Expressions (RegEx)

Logistic Regression

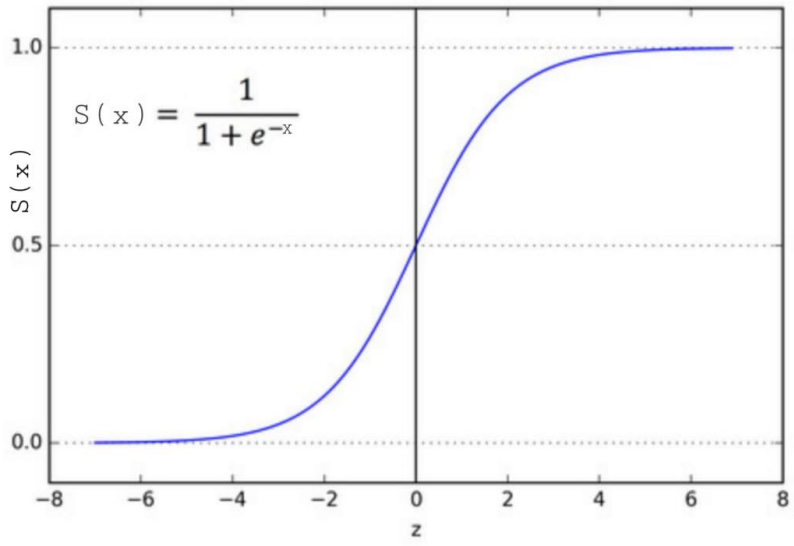
Logistic Regression is a Machine Learning method that is used to solve classification issues. It is a predictive analytic technique that is based on the probability idea. The classification algorithm Logistic Regression is used to predict the likelihood of a categorical dependent variable. The dependant variable in logistic regression is a binary variable with data coded as 1 (yes, True, normal, success, etc.) or 0 (no, False, abnormal, failure, etc.).

"Sigmoid function" or "logistic function"



The mathematically sigmoid function can be,

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

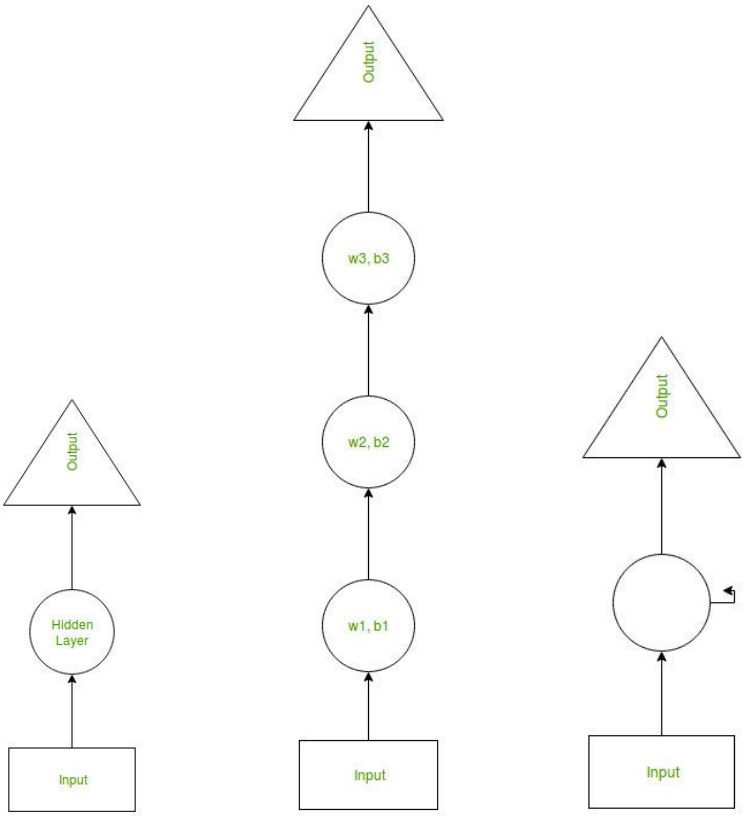


Padding Input Images

Padding is simply a process of adding layers of zeros to our input images so as to avoid the problems mentioned above.

Recurrent Neural Network

Recurrent Neural Network(RNN) are a type of Neural Network where the **output from previous step are fed as input to the current step**. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is **Hidden state**, which remembers some information about a sequence.



RNN have a **"memory"** which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

Advantages of Recurrent Neural Network

1. An RNN remembers each and every information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
2. Recurrent neural network are even used with convolutional layers to extend the effective pixel neighborhood.

Disadvantages of Recurrent Neural Network

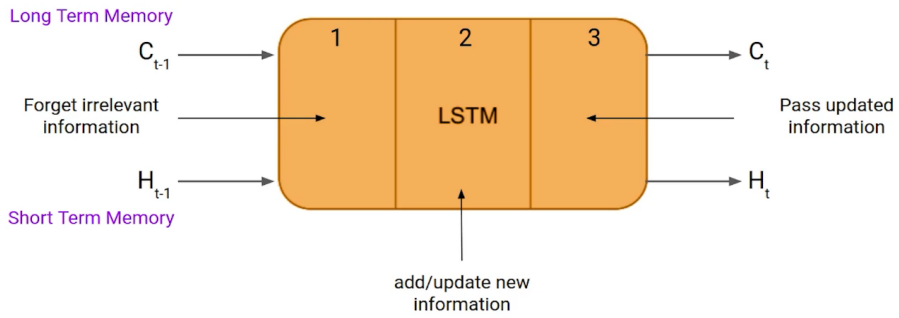
1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or relu as an activation function.

Long Short Term Memory

Long Short Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. LSTM was designed by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give an efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on the basis of time-series data.

Long Short Term Memory Network is an advanced RNN, a sequential network, that allows information to persist. It is capable of handling the vanishing gradient problem faced by RNN. A recurrent neural network is also known as RNN is used for persistent memory.

Let's say while watching a video you remember the previous scene or while reading a book you know what happened in the earlier chapter. Similarly RNNs work, they remember the previous information and use it for processing the current input. The shortcoming of RNN is, they can not remember Long term dependencies due to vanishing gradient. LSTMs are explicitly designed to avoid long-term dependency problems.



Texts_to_sequences

texts_to_sequences method **helps in converting tokens of text corpus into a sequence of integers**.

Adam Optimizer

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm.

StandardScaler follows **Standard Normal Distribution (SND)**. Therefore, it makes *mean = 0* and scales the data to unit variance.

What is hyperparameter optimization?

Before I define hyperparameter optimization, you need to understand what a hyperparameter is.

In short, hyperparameters are different parameter values that are used to control the learning process and have a significant effect on the performance of machine learning models.

An example of hyperparameters in the Random Forest algorithm is the number of estimators (*n_estimators*), maximum depth (*max_depth*), and criterion. These parameters are **tunable** and can directly affect how well a model trains.
So then **hyperparameter optimization** is the process of finding the right combination of hyperparameter values to achieve maximum performance on the data in a reasonable amount of time.
This process plays a vital role in the prediction accuracy of a machine learning algorithm.
Therefore Hyperparameter optimization is considered the **trickiest** part of building machine learning models.
Most of these machine learning algorithms come with the default values of their hyperparameters. But the default values do not always perform well on different types of Machine Learning projects. This is why you need to optimize them in order to get the right combination that will give you the best performance.

One-Hot Encoding

For categorical variables where no such ordinal relationship exists, the integer encoding is not enough.

In fact, using this encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

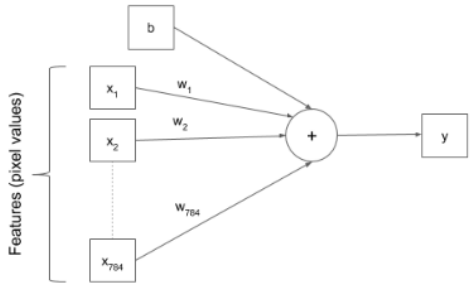
In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value.

In the "*color*" variable example, there are 3 categories and therefore 3 binary variables are needed. A "1" value is placed in the binary variable for the color and "0" values for the other colors.

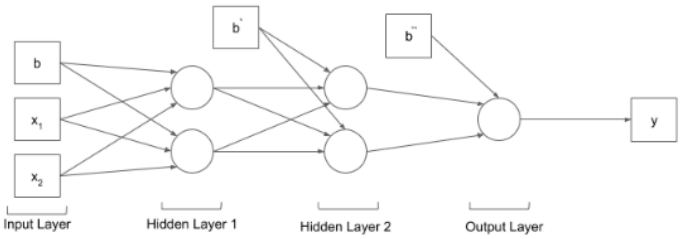
For example:

```
red,    green,  blue
1,      0,      0
0,      1,      0
0,      0,      1
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```



Neural Networks



Neural Networks

Activation Functions

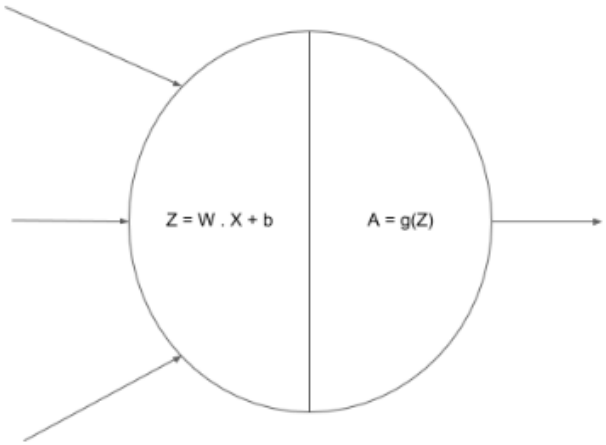
The first step in the node is the linear sum of the inputs:

$$Z = W.X + b$$

The second step in the node is the activation function output:

$$A = f(Z)$$

Graphical representation of a node where the two operations are performed:



Activation functions