1.1. Theoretical Task

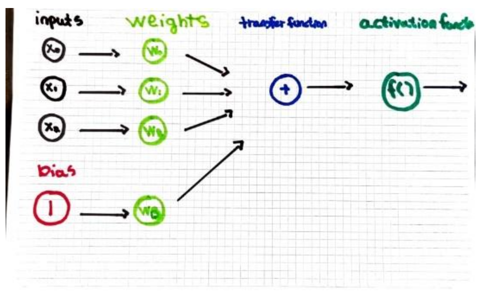
The human brain as far as we know is the most complex structure in the universe. We can use common sense, imagination and get inspired. Computers are getting smarter every day, they can do enormous calculations in a matter of seconds, but our brains are still a step ahead when it comes to interpret the world. We can use common sense, imagination to understand and solve problems in the real world. This is where Artificial Neural Network comes in (ANN). It is inspired by how our brains are structed hence the name. In a nutshell we can say the function of ANN is to make computers think and reason more like humans.

1)

An artificial neural network works by trying to mimic the why the network of neurons in our brain functions. This is achieved by programming regular computers to behave as though they are interconnected brain cells. By using different layers of mathematical processing, the computer can make sense of the data it is fed. A typical artificial neural network has anywhere between a few dozens to millions of artificial "neural" called units that has been arranged in a series of layers. When the first input layers receive information, this data will go through one or more hidden units. It is the hidden unit's job to transform the data to something the output unit can read. Most neural networks are connected from one layer to another. These connections are weighted. It is the weights job to assess the importance of each input. The higher the number of weights the greater influence one unit has another. It is the same way a human brain works. From here the data is passed on to the transfer function.

The transfer function in different from the components since it takes multiple inputs. This means that the job of the transfer functions is to combine multiple inputs into one output value that can be applied activation function. An activations functions transform the output from the transfer function into a value that dramatize the input. We will go into more depth in the following question.

One thin that is very important to mention is the Bias node. We can thin of this as a node that is "always on". This means that its value is set to 1 without the regard for the data in each pattern. The reason for this is that if a neural network does not have a bias node in each layer it will not be able to produce output in the next layer that differs from 0. The purpose is to allow the value before the activation function to be moved either up or down



regardless of the inputs them self

2)

We can loosely say there are four common types of Neural networks layers.

Fully Connected Layer

As the name implies, this kind of neural networks connect every unit in one layer t the nest layer. Fully connected layers are very common and can be found in many different types of neural networks such as standard neural networks to convolution neural networks (CNN).[1] One of the main drawbacks of this kind of neural networks is that since every layer is fully connected it can be become computationally expensive as their input grows. This can result in a combinational explosion of vector operations that can lead to potentially por scalability. Therefore, the most common use case for this in neural networks is tasks like classifying image data.

Convolution Layer

In simple terms convolution is the simple application of a filter to an input that results in an activation. Filter (also known as kernel) is a set of n-dimensional weights that are multiplied against an input. [2] The filter describes the probabilities that a pattern emerges in a given feature. The result of this is that the number of filters weights are smaller than the input and the multiplication process is performed in patches of the image that matches the filters.

[1] https://en.wikipedia.org/wiki/Convolutional_neural_network

[2] https://towardsdatascience.com/four-common-types-of-neural-network-layers-c0d3bb2a966c

Convolutional neural networks are most often used for vision problems where a computer must work with an image to extract features and or patterns.

Deconvolution Layer

Deconvolution as the name implies is all about reversing the effect of convolution. It is a transposed process that effectively up samples data to a higher resolution. In transposed convolution the output is larger the input. This is most often used to up sample images.

Recurrent Layer

Contrary to feed- forward neural network the information in Recurrent Neural Networks (RNN) cycles through a loop. This means that when it decides, it takes in account the currant input and what it has learned from previous inputs. As a result of this nature the recurrent layers are provided with a memory to maintain a state across iterations. The most common use case for this in translating text from one language to another, generating text and classifying sentiments into positive or negative. Both google assistant and Apple's Siri uses RNN.

3) Activation function

In simple terms activation function is a function that is added into ANN to the assist the network learns complex patterns in the data. If we use our brain as an example, the activation function is in the end deciding what is to be fired to the next neuron or next unit if we think of it as ANN. So, the activation functions take the output signal from the neuron or unit and converts it into some form that can be taken as an input to the next unit. A neural network without an activation is no good since it would basically just be a linear regression model. This is the reason we use a nonlinear transformation to the in put the input of the unit and this non-linearity in the network is introduced by an activation function. [3]

There are a lot of different and popular types of activation functions. Here are some of them.

Binary Step function

[3] https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/

Binary step functions use a threshold-based classifier which simply means whether a unit should be activated or not. If not, the output would be hidden for the next hidden layer. The binary step function can be used as an activation function while creating a binary classifier, but this is a very simple classifier. As a result of this the binary step functions is not suited for use where there are multiple classes in the target variable.

Sigmoid function

The Sigmoid function is one of the most common non-linear activation functions used. Unlike binary step function the Sigmoid function as mentioned above is a non-linear function, this means that when we have multiple units having sigmoid function as the activation function, the output will also be non-linear.

ReLU (Rectified Linear Unit) function

Like the Sigmoid function, the ReLU function is also a non-linear activation function that has gained a lot of popularity lately. Especially in the deep learning domain. Unlike other activation functions ReLU does not activate all units at once. This means that a unit will be deactivated if the linear transformation in less than 0.

There are a lot of other activation functions such as Linear functions, Tanh and Swish. There is also other activation that are very similar or build on the activation functions we have already talk about. These would be Parameterised ReLU, Leaky ReLU, Exponential Linear Unit and so on.

4)

Regression tree

A decision tree is basically an algorithm of if-else statements that can be used to predict a outcome based on the data given. For example, if a person can ride a roller coaster at an amusement park based on there height.

Regression tree is a variant of a decision tree. The aim is to predict outcome we consider real numbers, such as the number of expected cases of the Omicron variant this winter. It does this by trying to determine the relations between one dependent variable and several independent variables that diverged from the initial data set.

Building a regression tree

When building a regression tree, it is done through a process know as binary recursive partitioning. [4] This is a process where the data are split into sub-populations and gets split into further smaller groups as we move up each branch. In the beginning the data set is grouped into one partition. Then the algorithm begins to sort the data into the two first branches. The algorithm selects the split which results the least sum of squared deviations from the mean in two separate branches. This is repeated to each new branch. This process repeats itself until each node reaches a user specific node size at this point it will become a terminal node. If the sum of squared deviations from the mean is equal to zero, then that node is considering a terminal node.

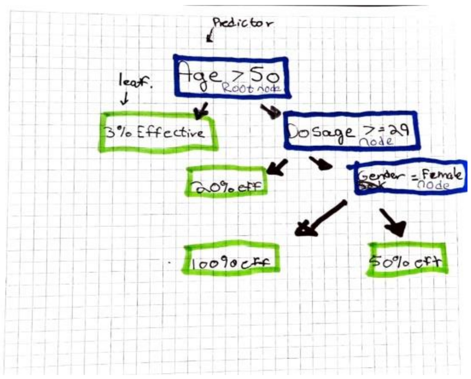
Example

Lets imagine we are creating a new drug to combat the newest variant of corona virus, Omicron. We do not know the optimal dosage of this drug, so we do a clinical trial with different dosages and measure how effective the dosages. Here we use a regression tree where each leaf represents a numerical value. The regression three is useful when there is more than one predictor.

Dosage mg	Age	Gender	Drug Effectiveness %
10	25	Male	98
20	78	Male	0
35	54	Female	100
5	11	Female	44
etc	etc	etc	etc

[4] https://en.wikipedia.org/wiki/Recursive_partitioning

Now we need to find the residual, the difference between the observed and predicted values and use it to quantify the quality of these predictions. We add all the squared residuals for every point. By calculating the average of two points and using it as new threshold and thereby getting new averages, residuals, and sum of squared residuals. We do this until we have calculated the sum of squared residuals for each threshold for each of the predictors. We pick the threshold from all the predictors with the minimum sum of squared residuals. This becomes the root of the tree. In this case it would be Age > 50. Then we grow the three by comparing the lowest sum of squared residuals from each predictor. When there is fewer than minimum numbers of observation usually 20 then that node becomes a leaf. If not, we keep splitting the remaining observation until we cannot split it anymore.



Pruning the Tree

As a result of the tree growing from the start data set, it can suffer from overfitting.[5] This occurs when a function is overly aligned to a limited set of data points. This can result in overly complex models to explain "quirks" in the data set. Because of this we need to prune the tree. The pruning process can be divided up in two types. The pre- and post-post pruning.

References:

https://towardsdatascience.com/artificial-neural-networks-for-total-beginners-d8cd07abaae4 (Rich Stureborg, 2019)

https://towardsdatascience.com/four-common-types-of-neural-network-layers-c0d3bb2a966c (Martin Isaksson, 2020)

5 https://www.investopedia.com/terms/o/overfitting.asp#:~:text=Overfitting%20is%20a%20modeling%20error,limited%20set%20of%20data%20points.&text=Thus%2C%20attempting%20to%20make%20the,and%20reduce%20its%20predictive%20power.

https://builtin.com/data-science/recurrent-neural-networks-and-lstm (Niklas Donges, 2021)

https://medium.com/@marsxiang/convolutions-transposed-and-deconvolution-6430c358a5b6 (Mars Xiang, 2020)

https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/ (Jason **Brownlee, 2019)**

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Sumit Saha, 2018)

https://builtin.com/data-science/regression-tree (Kushal Vala, 2021)

https://www.investopedia.com/terms/o/overfitting.asp#:~:text=Overfitting%20is%20a%20modeling%20error,limited%20set%20of%20data%20points.&text=Thus%2C%20attempting%20to%20make%20the,and%20reduce%20its%20predictive%20power. (Alexandra Twin, 2021)

https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/ (Dishashree26 Gupta, 2020)

Support Vector Machine Classification Report

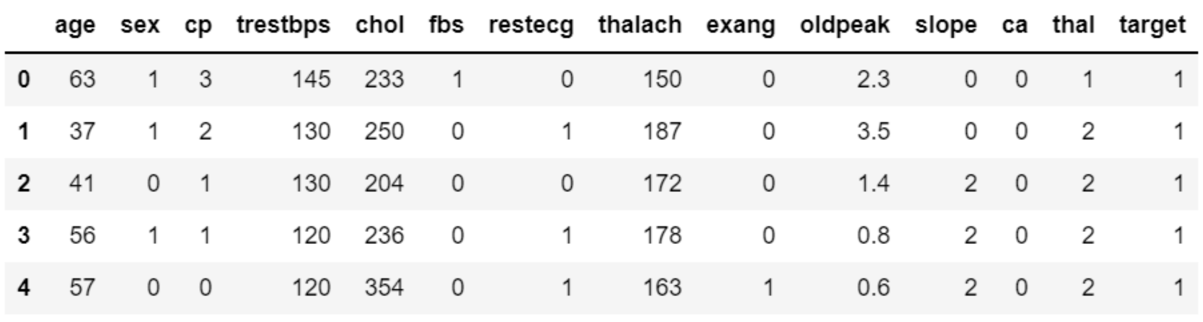
Introduction

Report is part of the exam for ML and NLP course, there are 5 questions in the task. There are three practical tasks and two theoretical tasks that will be answered and highlighted in this report. The original support vector machine came to life already in 1963, and was further developed in 1992 with Bernard Boser, Isabelle Guyon and Vladimir Vapnik where they wanted to introduce the kernel trick which allows for high-dimensional, implicit feature space and cheaper computations. Support Vector Machine is a supervised and simple machine learning algorithm, where you can either analyze data for classification or regression analytics. Let us see how well it manages our classification problem.

(Guyon, 2016)

Data

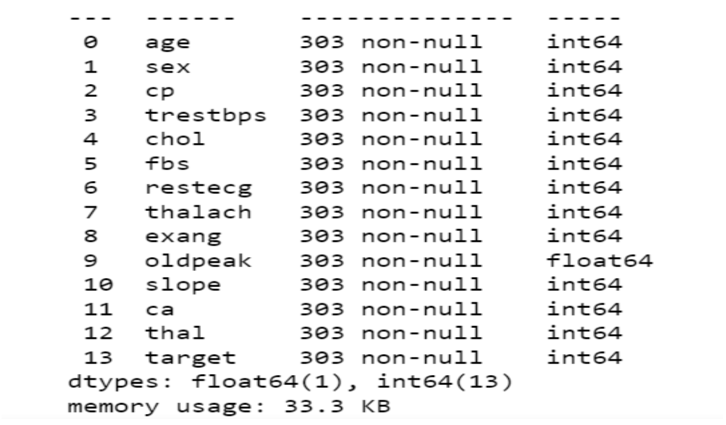
The data we will be using for this task is Heart Disease UCI, obtained from Kaggle.com. In context the original database contains up to seventy-six attributes but has been cut down to a subset of fourteen. (Kaggle, 2018)



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Data Preprocessing

The first steps in our preprocessing is to locate any outliers, missing values and see if there is any additional code we need to implement in order to read the dataset correctly.



```
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

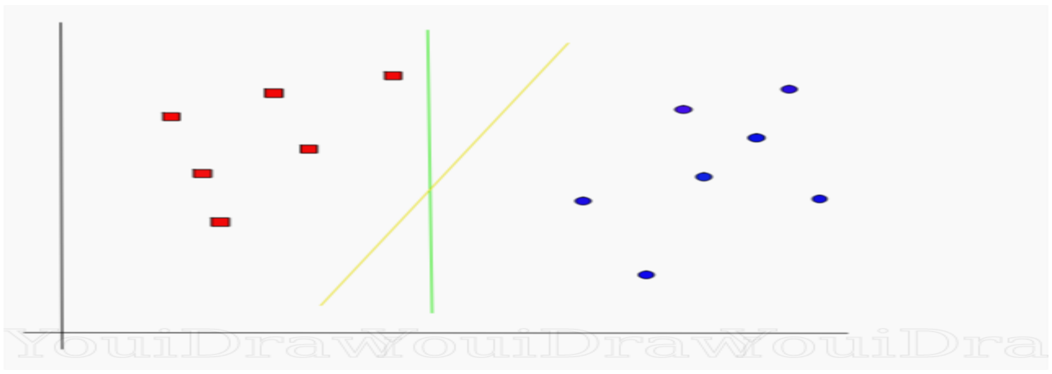
Running df.head() after importing our dataset will give us a nice overview of what kind of data we are dealing with. As we can see in the picture above, we are dealing with mostly integers and non-null values which indicates that there is not a lot of preprocessing of the dataset to be done, in the case of categorical data we could have encoded categorical data into values but there is no need for that in our case.

We move on to splitting the dataset, allocating all columns to X and y to “target”. Target represents if the patient has heart disease or not. Importing train_test_split after and allocating 20 % to test and the remaining 80% to train.

The last step in our preprocessing part we introduce the StandardScaler.

Support Vector Machine

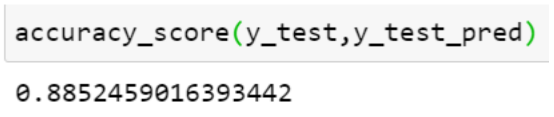
Support vector machine formally constructs a hyperplane in a high dimensional space and separates data into classes. A hyperplane, often referred to in geometry where it is a subspace of a dimension. In the picture below you can see the line separate data points, there are several ways to separate the points, but it is the Support Vector Machine's job to find the ideal and best line to draw even in more complex and not linear separable datasets. (pupale, 2018)



Implementing SVM to our Dataset.

We decided to go with a linear kernel on our first initial try. We performed a Grid Search to find the best parameters for our kernel with cross validation.

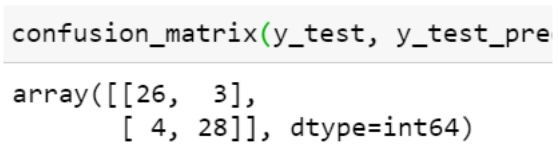
This is the accuracy result:



```
accuracy_score(y_test,y_test_pred)
```

```
0.8852459016393442
```

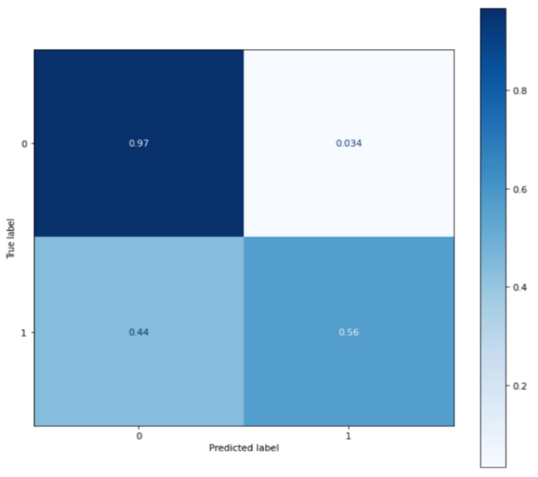
Confusion matrix:

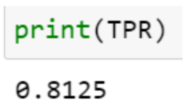


```
confusion_matrix(y_test, y_test_pre
```

```
array([[26,  3],
       [ 4, 28]], dtype=int64)
```

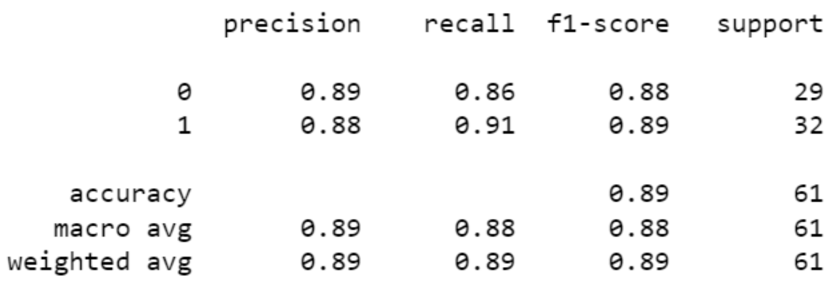
Confusion matrix display:





```
print(TPR)

0.8125
```



```
              precision    recall  f1-score   support

           0       0.89      0.86      0.88        29
           1       0.88      0.91      0.89        32

    accuracy                           0.89        61
   macro avg       0.89      0.88      0.88        61
weighted avg       0.89      0.89      0.89        61
```

We can see that the model we have went with gives us an accuracy of 88%, which is good. We have done Grid Search, to find the best parameters within 0.1, 1 and 100. What we could do to find further improvements is to try more parameters. Because our accuracy is at 88%, we can look to further improve but do not need. We could add more data or do better feature selection. Top rows in the above picture shows True positive and False positive, bottom represents False negative and True negative.

(Ragan, 2018)

What will cause the SVM to fail or not be efficient?

SVM classifier will fail when implementing the wrong kernel function, for example introducing a linear kernel to data that is not linear separable.

SVM classifier will fail if you do not put in the right key parameters, this will be fixed with a Grid Search, which will highlight the best parameters out of a set of selected parameters. It can also work with MKL. SVM manages small to mid-size datasets well but handling bigger datasets can be an issue. In our case, SWM works good on the linear kernel, which was randomly implemented, we do not have too big of a dataset to work with, and we have also run Grid Search/Hyper tuning to find the best of the three.

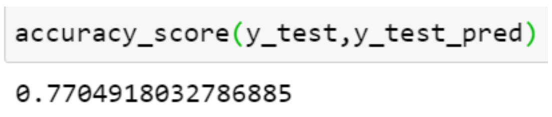
(Khoong, 2021)

RBF vs Polynomial vs Linear

In the last task we wanted to see if there was any difference between the three kernels we have used. We decided to go with RBF and Polynomial kernels compared to a linear kernel. The difference in accuracy may vary. For example, we may have not found the most optimal parameters for our kernels. We did run a Grid Search between the two with even more parameters than the linear, but they are still lacking compared to linear. We could also argue that the data we are working with are more suitable for a linear kernel, regarding where the datapoints are in the hyperplane.

(Ray, 2017)

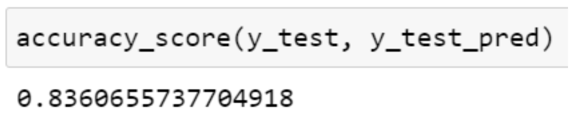
Polynomial accuracy:



```
accuracy_score(y_test,y_test_pred)
```

```
0.7704918032786885
```

RBF accuracy:



```
accuracy_score(y_test, y_test_pred)
```

```
0.8360655737704918
```

Conclusion and recommendations:

Support Vector Machine is a good machine learning algorithm that can be used for both classification and regression problems you might face, do note that it is mainly used for classification. Using SVM on larger datasets or datasets containing more noise where the data you want to target overlaps with the classes, you want to look for another machine learning algorithm, due to its time consummation with training. But if you are working with smaller datasets, SVM is a great choice to go with. Obtaining good results with a pretty simple algorithm and great memory efficiency due to its support vectors.

(Ray, 2017)

Referanser

Guyon, I. (2016). *KD Nuggets* . Hentet fra KD Nuggets : https://www.kdnuggets.com/2016/07/guyon-data-mining-history-svm-support-vector-machines.html

Kaggle. (2018). *Kaggle*. Hentet fra Datasets: https://www.kaggle.com/ronitf/heart-disease-uci

pupale, R. (2018, June 16). *Towards Data Science* . Hentet fra Support Vector Machines - An overview : https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

Ragan, A. (2018, October 10). *Towards Data Science* . Hentet fra Taking the confusion out of confusion matrices : https://towardsdatascience.com/taking-the-confusion-out-of-confusion-matrices-c1ce054b3d3e

Ray, S. (2017, September 13). *Analytics Vidhya*. Hentet fra https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

Wikepedia. (2021, December 6). *Wikepedia*. Hentet fra Wikepedia: https://en.wikipedia.org/wiki/Support-vector_machine

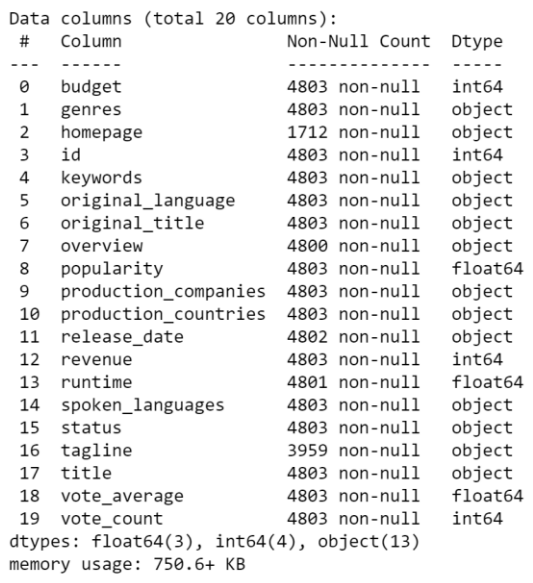
Wikepedia. (2021, October 29). *Wikepedia*. Hentet fra Wikepedia: https://en.wikipedia.org/wiki/Kernel_method

Wikepedia. (2021, October 30). *Wikepedia*. Hentet fra Hyperplane: https://en.wikipedia.org/wiki/Hyperplane

2.1. Practical Task: Text processing, feature extraction and representation by using both TF and TF-IDF schemes

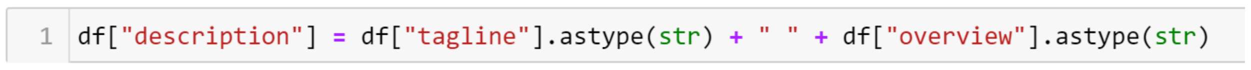
1. Data Preprocessing

The data is about film reviews, maybe it can be used to make a recommendation engine for viewers. The dataframe includes 4803 rows and 21 columns. There are 3091 missing values in column homepage, 844 missing values in tagline.



```
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   budget                4803 non-null   int64
 1   genres                4803 non-null   object
 2   homepage              1712 non-null   object
 3   id                    4803 non-null   int64
 4   keywords              4803 non-null   object
 5   original_language     4803 non-null   object
 6   original_title        4803 non-null   object
 7   overview              4800 non-null   object
 8   popularity            4803 non-null   float64
 9   production_companies  4803 non-null   object
 10  production_countries  4803 non-null   object
 11  release_date          4802 non-null   object
 12  revenue               4803 non-null   int64
 13  runtime               4801 non-null   float64
 14  spoken_languages      4803 non-null   object
 15  status                4803 non-null   object
 16  tagline               3959 non-null   object
 17  title                 4803 non-null   object
 18  vote_average          4803 non-null   float64
 19  vote_count            4803 non-null   int64
dtypes: float64(3), int64(4), object(13)
memory usage: 750.6+ KB
```

New column description was created by concatenating the strings from two columns: tagline and overview as followed:



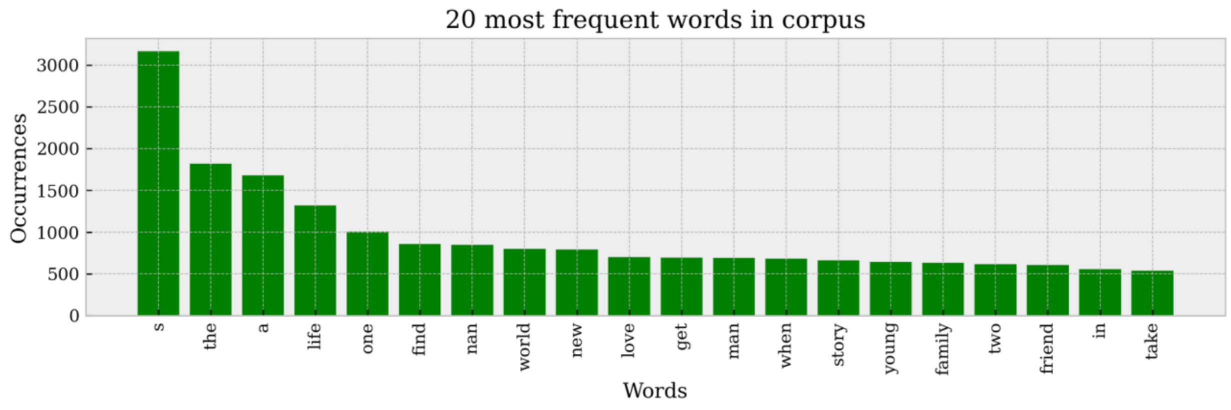
```
df["description"] = df["tagline"].astype(str) + " " + df["overview"].astype(str)
```

2. Text processing

The first step of text analysis is tokenization. Tokenization is a fundamental of natural language processing (NLP) and it is described separating texts into smaller pieces called tokens. These tokens can be either words, numbers, characters, or other types of word forms but in our study, we prefer to eliminate everything except words and numbers. Basic preprocessing steps were applied to the texts along with token creation by ‘preprocessing(corpus)’ function, which involves token lemmatization to reduce token to their base form, token.strip() to remove space, token.lower() to make all the token to lowercase and finally separate stop words from tokens.

Bag of word (bow) representation and Time Frequency (tf)

Texts cannot be implemented directly into machine learning models. Bow is one of the necessary preprocessing steps in NLP which turns regular texts to ‘bag-of-words’. Bow has a dictionary like structure, and it has basically two information: (a) words in the raw text and (b) frequency of occurrence of words. Bows are created in three steps in this study. First a ‘corpus’, the combination of all raw texts, was created.



Tf vectors represent absolute numbers which a word appears in a bow and this is not very ideal. In a tf vector, it is not easy to understand weight of a word in a text. It is not the same appearing 5 times in a text with 100 words and another one with 200 words.

TF and TF-IDF representation on 'description'

TF-IDF model is a quick way to acquire information about texts and documents which allows get meaning from texts and compare documents in a mathematical way. It is very easy to implement the code and the method is not computationally expensive. Besides the advantages it also has some disadvantages. It computes document (text) similarity based on only word-count (bag-of-words) and it does not take into account word's position in the text, semantics and co-occurrences (Ahuja, 2020). TF-IDF method based on word count gives enough information about text similarity which is not always the case. It may be slow to implement this method for large text.

TF means term frequency and IDF denotes inverse document frequency. The more times a word appears in the document, the TF (and hence TF-IDF) will go up. The number of documents that contain that word goes up, the IDF (and hence the TF-IDF) for that word will go down.

tf(t, d) = count of word t in d / number of words in d
idf(t, D) = log(number of documents/number of documents containing the word t)
tfidf(t, d, D) = tf(t, d) * idf(t, D).

The main function of TF-IDF is to increase value of tokens which are important in a text or document (Chakravarthy, 2020). TF-IDF model is a quick way to acquire information about texts and documents which allows get meaning from texts and compare documents in a mathematical way. It is very easy to implement the code and the method is not computationally expensive. Besides the advantages it also has some disadvantages. It computes document (text) similarity based on only word-count (bag-of-words) and it does not take into account word’s position in the text, semantics and co-occurrences (Ahuja, 2020). TF-IDF method based on word count gives enough information about text similarity which is not always the case. It may be slow to implement this method for large text.

2.2. Practical Task: Topic modelling

Truncated SVD

LSA is recommended in the course to be our first choice for topic modelling, semantic search, or content-based recommendation engines. LSA (Latent semantic analysis) uses SVD to find the combination of words, that are responsible, together, for the biggest variation in the data. SVD decomposes a matrix into three square matrices, one of which is diagonal.

We use scikit-learns’ TF-IDF vectorizer to take corpus and convert each document into a sparse matrix of TFIDF features

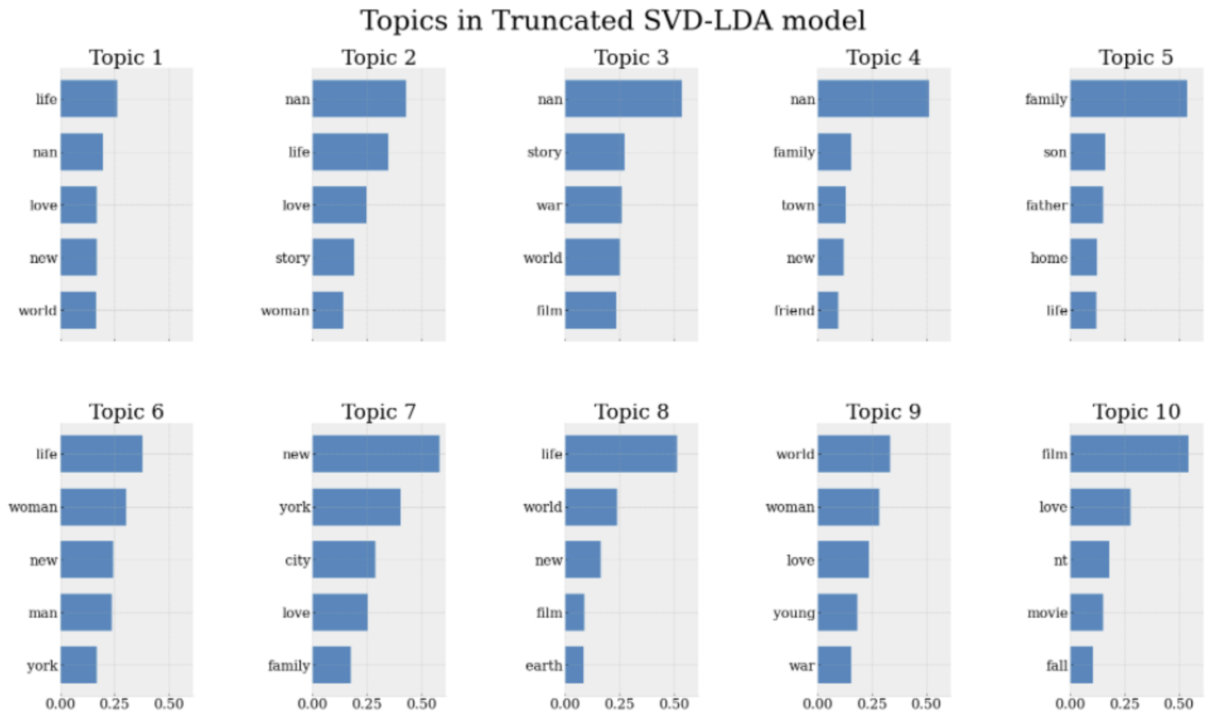
X is input matrix where m is the number of document (n) we have, and m is the number of terms. We decompose X into three matrices called U, S and T. When we do the decomposition, we have to pick a value p, that’s how many topics or concepts we keep.

$$\mathbf{X} \dashrightarrow \mathbf{U}_{mxp}\mathbf{S}_{pxp}\mathbf{V}_{pxn}^{T}$$

m: number of terms in vocabulary
n: number of documents
p: number of topics (same as the number of words)

U is a mxp matrix. The rows will be documents and the columns will be ‘concepts’
S is a kxk diagonal matrix. The elements will be the amount of variation captured from each concept.

V is a mxk (mind the transpose) matrix. The rows is the terms and the columns is the concepts.

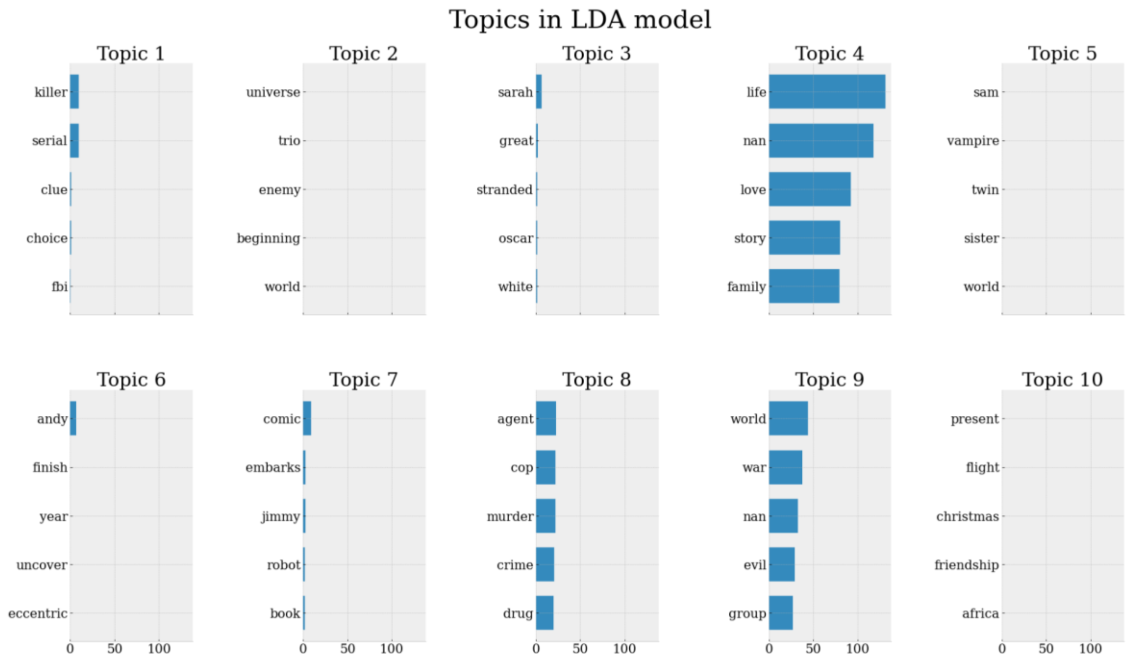


Here we see that the first topic is about man love life, second about woman love life story, third is about world war film story, forth is about new family friend in town, fifth is about family with father and son osv.

LDiA/LDA (Laten Dirichlet Allocation)

In some situation, LDA can give slightly better results than LSA. LDA is introduced by David Blei, Andrew Ng and Michael O. Jordan in 2003 (Edward, 2018). LDA is the most common algorithm for topic modelling. LDA assumes a Dirichlet distribution of word frequencies. It is more precise about the statistics of allocation words to topics than the linear math of LSA. LDA assumes that each document is a mixture (linear combination) of some arbitrary number of topics. It assumes that each topic can be represented by a distribution of words (term frequencies). The probability or weight for each of these topics within a document, as well as the probability of a word being assigned to a topic, is assumed to start with a Dirichlet probability distribution.

Both SVD and LDA use Bag-of-words as input matrix. However, when we use SVD, it is difficult to determine the optimal number of dimensions. In details, low dimension consumes less resources but we may not able to distinguish opposite meaning words while high dimension overcome it but consuming more resource (Edward, 2018).



Here we see that the first topic seems to be about serial killer with clue. The second topic is about beginning of new world and universe, third is about great white oscar, fourth is about man love story, fifth is vampire twin sister osv.

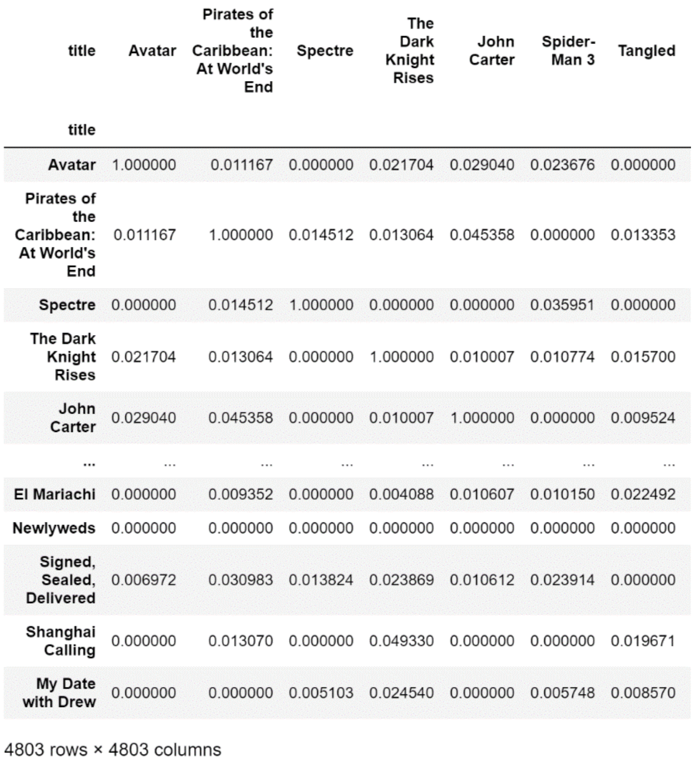
Word2vec

Word2Vec takes into account not only words but also context and semantics. Word2vec algorithm uses neural network model to learn associations in a large corpus (Wikipedia 2021). It can detect synonyms and suggest words for partial sentences. It uses word-embedding very efficiently. Therefore, it is used very efficiently by companies like Ali express, AIR BNB, Spotify, etc (Alammar, 2019).

Another efficient alternative NLP model to TF-IDF is BERT (Bidirectional Encoder Representations from Transformers). This method is released in 2018 and stirred NLP community by its highly accurate results in different NLP tasks (Horev, 2018). BERT uses a transformer, an attention mechanism, that learns contextual relationships between words.

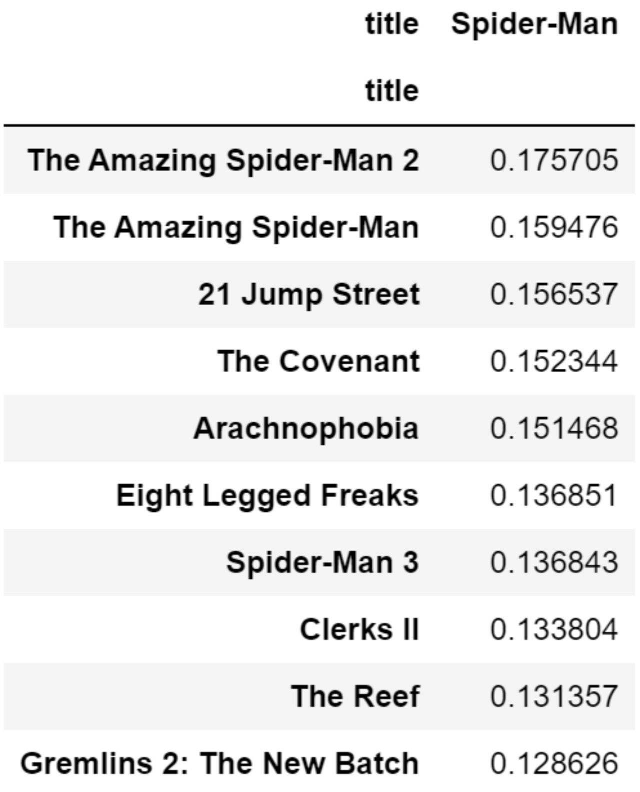
The transformer has two mechanisms – an encoder which reads texts and a decoder which produces predictions for the text. Directional models read text sequentially, from left to right or right to left. But BERT reads text in both directions; left-to-right and right-to-left. This specialty allows BERT to learn better from words surroundings, from both left and right. One of the most used methods in both industrial and research purposes is ELMo. ELMo models both complex characteristics of word use (syntax, semantics, etc.) and how words are used in linguistic contexts. The internal states of bidirectional language model (biLM), pre-trained in corpus, is used to create word vectors. These vectors can be added to existing model and improve results significantly in different NLP works such as question answering or sentiment analysis (Peters et al. 2018).

2.3. Analysis Task: Searching for similar movies



title	Avatar	Pirates of the Caribbean: At World's End	Spectre	The Dark Knight Rises	John Carter	Spider-Man 3	Tangled
title							
Avatar	1.000000	0.011167	0.000000	0.021704	0.029040	0.023676	0.000000
Pirates of the Caribbean: At World's End	0.011167	1.000000	0.014512	0.013064	0.045358	0.000000	0.013353
Spectre	0.000000	0.014512	1.000000	0.000000	0.000000	0.035951	0.000000
The Dark Knight Rises	0.021704	0.013064	0.000000	1.000000	0.010007	0.010774	0.015700
John Carter	0.029040	0.045358	0.000000	0.010007	1.000000	0.000000	0.009524
...	...	...	...	...	...	...	...
El Mariachi	0.000000	0.009352	0.000000	0.004088	0.010607	0.010150	0.022492
Newlyweds	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Signed, Sealed, Delivered	0.006972	0.030983	0.013824	0.023869	0.010612	0.023914	0.000000
Shanghai Calling	0.000000	0.013070	0.000000	0.049330	0.000000	0.000000	0.019671
My Date with Drew	0.000000	0.000000	0.005103	0.024540	0.000000	0.005748	0.008570

4803 rows × 4803 columns



title	Spider-Man
title	
The Amazing Spider-Man 2	0.175705
The Amazing Spider-Man	0.159476
21 Jump Street	0.156537
The Covenant	0.152344
Arachnophobia	0.151468
Eight Legged Freaks	0.136851
Spider-Man 3	0.136843
Clerks II	0.133804
The Reef	0.131357
Gremlins 2: The New Batch	0.128626

Conclusions and Recommendations

References

Jay Alammar (March 27, 2019). The illustrated Word2vec. https://jalammar.github.io/illustratedword2vec/

Madhu Sanjeevi (2017). Chapter 9.1: NLP - Word vectors. https://medium.com/deep-math-machine-learning-ai/chapter-9-1-nlp-word-vectors-d51bff9628c1.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer. Deep contextualized word representations.

NAACL 2018. Pallavi Ahuja, (2020). How Good (or Bad) is Traditional TF-IDF Text Mining Technique? https://medium.com/analytics-vidhya/how-good-or-bad-is-traditional-tf-idf-text-mining-technique-304aec920009

Rani Horev (November 10, 2018). Best explained: State of the art language model for NLP. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

Srinivas Chakravarthy (2020). Simple Word Embedding for Natural Language Processing. https://towardsdatascience.com/simple-word-embedding-for-natural-language-processing5484eeb05c06.

Wikipedia (2021). Word2vec. https://en.wikipedia.org/wiki/Word2vec. Accessed October 29, 2021

https://www.youtube.com/watch?v=BJ0MnawUpaU

Edward (2018), 2 latent methods for dimension reduction and topic modeling | by Edward Ma | Towards Data Science