Classification models benchmarking

Wajih Arfaoui

Logistic Regression

1. Main Objective

Logistic regression is a kind of parametric classification model. It uses a linear combination of features to come out with a probability to assign two values 0 and 1 fail or true and false to the response variable.
The middle value of probabilities is considered as threshold to establish what belong to the class 1 and to the class 0. In a general note, if the probability is greater that 0.5, then the observation belongs to class 1, otherwise it is belongs to class 0.

2. Logit Function

Logistic regression is expressed as:

$$log(\frac{p(X)}{1-p(X)}) = \beta_0 + \beta_1 X$$

Where the left-hand side is called the *log-odds* and $\frac{p(X)}{1-p(X)}$ is called *odds*, and it tells about the probability of success to probability of failure.

When taking the inverse of the logit function we will get:

$$p(X) = (\frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}})$$

This function is the **Sigmoid** function and it creates the *S-shaped* curve, and returns a probability value between 0 and 1.

3. Maximum Likelihood Estimation

Since **Maximum Likelihood** is the more general method of non-linear least squares and it has a better statistical properties, it is used to fit the logistic regression model.
The maximum likelihood estimation defines the coefficients for which the the probability of getting the observed data is maximized.

The *likelihood* function formalizing the stated intuition is:

$$L(\beta, y) = \prod_{i=1}^{N} (\frac{\pi_i}{1-\pi_i})^y (1-\pi_i)$$

$$\text{for } y_i = [0, 1]$$

$$\pi_i \text{ is the probability of success if } y_i \text{ belongs to class 1}$$

In order, to determine the parameters' values, we apply $log()$ to the likelihood function, since it does not change its initiatl propoerties. Then we apply **iterative** optimisation techniques such as **Gradient Descent**.

4. Pros & Cons

Pros	Cons
- Logistic Regression is easy to interpret and very efficient to train	- Logistic Regression may lead to overfitting when the number of features is greater than the number of observations
- In addition to providing coefficients' sizes, it tells the direction of association	- It assumes linearity between the dependent variable and the target
- It doesn't need hyperparameter tuning	- It is not considered as a very powerful algorithm and can be easily outperformed by other algorithms

Linear Discriminant Analysis

1. Main Objective

The aim of LDA is to maximize the **between-class** variance and **minimize the within-class** variance through a linear discriminant function. It assumes that all classes are linearly separable, and the data in each class is described by a **Gaussian** probability density function which means that it has a bell-curve shape when plotted.

2. Linear Descriminante function

Since the LDA uses the Bayes' Theorem to make predictions based on the probability that the observation x belongs to each class. The class having the highest probability is designated as the output class, and then prediction is made by the LDA.
The *Bayes' theorem states that:

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{k} \pi_l f_l(x)}$$

Where x= input

k= output class

π_k = prior probability that an observation belongs to class k

$f_k(x)$ = estimated probability of x belonging to class k

Supposing that $f_k(x)$ follows a Gaussian Distribution, it takes the form:

$$f_k(x) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2)$$

After plugging this into the probability equation and taking the log of it, we get the following discriminant function:

$$\delta_k(x) = x.\frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

3. LDA assumptions

As already mentioned, LDA makes two important assumptions about the data:
- Each data variable is bell curve shaped
- The values of each variable vary around the mean by the same amount on the average.

Based on that, LDA method approximates the Bayes classifier by plugging estimates for π_k, μ_k and σ^2 into the linear discriminant function.
The following estimates ares used:

$$\hat{\mu_k} = \frac{1}{n_k} \sum_{i:y=k} x_i$$

$$\hat{\sigma}^2 = \frac{1}{n-k} \sum_{k=1}^{k} \sum_{i:y=k} (x_i - \hat{\mu_k})^2$$

$$\hat{\pi}_k = n_k/n$$

Where $\hat{\pi}_k$ = variance across all inputs x

n = number of instances

k = number of classes

$\hat{\mu_k}$ = mean for input x

4. Pros & Cons

Pros	Cons
- LDA is simple to implement and the classification is robust - It uses information from both the features to create a new axis which in turn minimizes the variance and maximizes the class distance of the two variables	- LDA's linear decision boundaries may not adequately separate the classes - It requires normal distribution assumption on features - It has a large time complexity

K-Nearest Neighbors

1. Main Objective

K-nearest neighbors is a supervised machine learning algorithm that is used for both, regression and classification. It predicts the correct class for the test observation by measuring the distance between it and all the training data points. Then it tries to find the **nearest neighbors** by ranking points by increasing distance,and finally vote for the most frequent label to be assigned to the test data point, after selecting the specified number k of nearest neighbors to consider.

2. Calculating distance

The first step in the KNN application is to calculate the distance between the data point we want to classify and our training data points.
To do this, we need to choose one of the various methods used to measure the distance such as:

- **Euclidean Distance**: It is calculated as the square root of the sum of the squared differences between the test data point t and a train data point x.

$$\sqrt{\sum_{i=1}^{k}(t_i - x_i)^2}$$

- **Manhattan Distance**: It calculated the distance between two points in a N dimensional vector space by summing the absolute difference between the measures in all dimensions of two points.

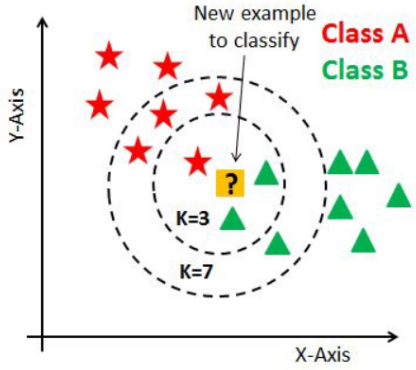
$$\sum_{i=1}^{k}|t_i - x_i|$$

The two stated methods above are applied for continuous variables, while in the case of categorical variables, we opt for:

- **Hamming Distance**: It counts the number of times the coordinates in two categorical vectors differ. It is mainly used when you one-hot encode your data and need to find distances between the two binary vectors.

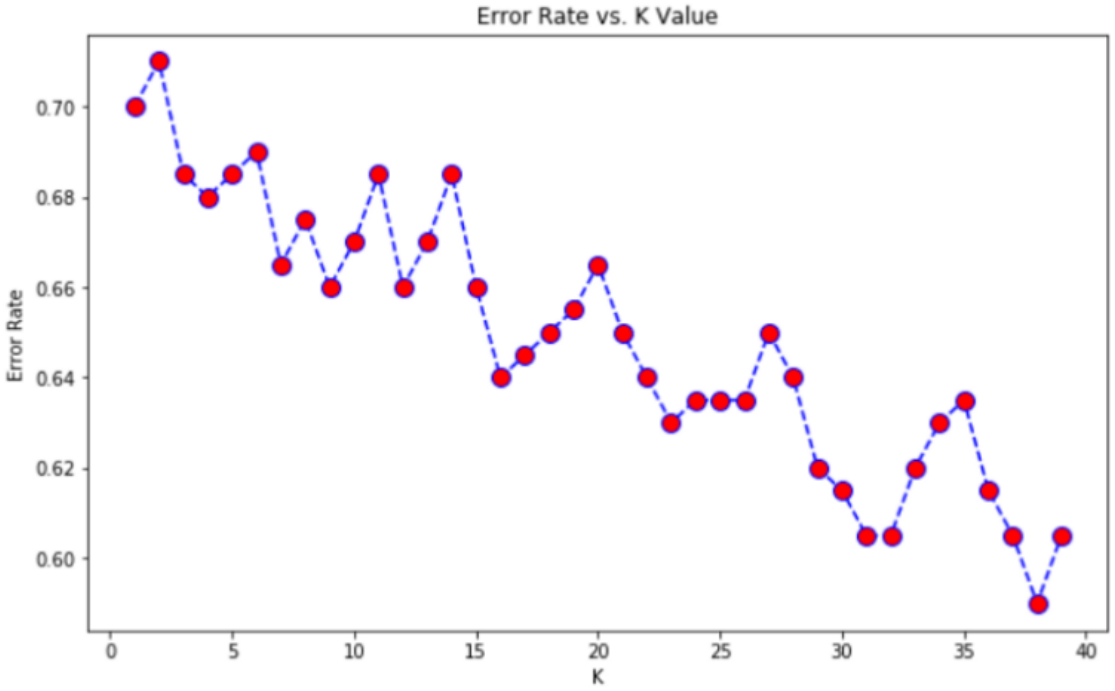
3. Choosing the right value for K

As **K** value indicates the number of nearest neighbors, we need to look for the optimal value to ensure we get the most accurate classification.
From the photo below, if we choose K=3, we predict that test observations belongs to class B, but it belongs to class A if we choose K to be 7.



So, to select the K that is right for the data, we don't have pre-defined statistical methods to find the most favorable one, but instead we need to run the KNN algorithm multiple times with a different K each time. Then, we choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before as show in the plot below.

```
Minimum error:- 0.59 at K = 37
```



4. Pros & Cons

Pros	Cons
- It is simple to implement since it has only one parameter k	- It is a lazy learner, it takes longer time for inference than training
- It allows dealing with complex objects, such as time series, graphs and any distance metric even defined by the user	- It is prone to overfitting as it is a distance based approach and can be affected by outliers
- It is versatile, since it can be used for classification, regression, recommendations etc.	- It has the curse of dimensionality

Classification Decision Trees

1. Main Objective

Classification trees is used in order to create a training model that leads to a decision about the class of an object by learning simple decision rules inferred from the training dataset.
In Decision trees, in order to predict a class label for the test observation, we start from the root of the tree, then we compare the values of the root attribute to the observation that we want to classify attribute. Once the comparison is done, we follow the branch corresponding to that value and we move on to the next node.

2. Recursive Binary Splitting

The decision trees are built using a heuristic method called recursive binary splitting. This approach considers all the features and it splits the data into subsets that are themselves splitted into smaller subsets.

After testing all our candidates' splits in each level, we calculate how much accuracy each split will cost us, using a function and we choose the one with that costs the least.

Since we are basing our judgement on how homogeneous data is within each subset, we use the following cost function (Gini Index):

$$\mathrm{G} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

where $\hat{p}_{mk}$ represents the proportion of training observations in the m^{th} group that are from the k^{th} class. This measure gives an idea about how good the split is, by defining how mixed the response classes are in the groups created. According to the function above, we can notice that the Gini Index takes a small value when all of the $\hat{p}_{mk}$'s are close to 0 or 1. It is for this reason that Gini is refereed to as the measure of node **purity** and the smaller the Gini Index is the more pure the node is.

3. Pruning

We can improve the performance of the tree by **pruning** which means we remove the branches that make use of features with low importance. By doing so, we can reduce the complexity of the tree, and we increase its predictive power by reducing overfitting.

The classic way of pruning is to start at leaves and to remove each node with the most popular class in that leaf, and we keep this change if it doesn't negatively affect the accuracy. This approach is called **reduced error pruning**.
A more sophisticated approach can be also applied which is the **cost complexity pruning** that uses a learning parameter α to weigh whether nodes can be removed based on their sub-trees sizes.

4. Pros & Cons

Pros	Cons
- It is inexpensive to construct, and it is fast at classifiying unknown observations	- It is easy to overfit
- It is easy to explain and interpret	- Small changes in the training data can result in large changes to decision logic
- It can easily handle qualitative predictors without the need of creating dummy variables	- It doesn't have the same level of predictive accuracy as some of the classification approaches

Support Vector Machine

1. Main Objective

Support Vector Machine (SVM) is a supervised machine learning algorithm used mainly for binary classification problems, even though it can also be used for regression as well. The objective of SVM is to find a N-dimension hyperplane that distinctly classifies data points.
In order to find the most appropriate hyperplane, which means maximize the margin of the classifier, we rely on support vectors that are the closest data points to the hyperplane and can influence its position and orientation.

2. Objective function

The outliers caused by this noise can affect the positioning of the hyperplane, so to avoid this, we need to find a hyperplane that tolerates at a certain extent misclassification by relaxing the used constraint in linearity separable case:

$$z_i(w^T x_i + b) \geq 1$$

We insert a slack $s_i \geq 0$ for each observation x_i. This s_i indicates the deviation of x_i from the corresponding hyperplane for $i = 1, ..., n$, and we receive the following modified constraint:

$$z_i(w^T x_i + b) + s_i \geq 1$$

Hence, the sum of s_i represents the total deviation error which is considered as the upper bound for the total missclassified data points.
Simultaneously, we can add a hyperparameter $v > 0$ in order to regulate the influence of s_i, since it ensures achieving the best ratio between the margin with the highest width and the lowest number of misclassified data points.

Finally, the objective function for the SVM is represented as follows:

$$min_{(w,b,s)} v \sum_{i=1}^{n} s_i + \frac{1}{2}||w||^2$$

$$\text{s.to} : z_i(w^T x_i + b) + s_i \geq 1$$

$$x_i \geq 0, s_i \geq 0, b \geq 0 \text{ for } i = 1, ..., n$$

Where:
- v denotes the regularization hyperparameter.
- s_i denotes the distance of x_i from the separating hyperplane.
- b denotes a bias parameter
- $||w||^2 = w^T w$
- z_i denotes a variable that equals 1 if x_i belongs to class 1and -1 if x_i belongs to class 2.

3. Pros & Cons

Pros	Cons
- It is very effective when it comes to high dimensional spaces	- It can be quite costly computationally to train
- It is relatively memory efficient	- It does not perform very well when the data set has more noise
- It can be used to both classify data and also predict continuous numerical values	- It doesn't work that well with mid-size or large datasets

Benchmarking

1. Feature Selection

After reading the data, doing the cleaning, feature engineering, the scaling and separating the **Independant variables** from **Dependant variable**, we proceed to do the feature selection.

Since we have numerical input variables and a binary target variable used for classification, I used **ANOVA f-test** as a selection model, and I ended up having these 15 features with the highest variances:

LIMIT_BAL	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	Male	High_School	Other_edu	Uni	unknown	married	single	Pay_AMT_Total

2. Cross validation & Sampling

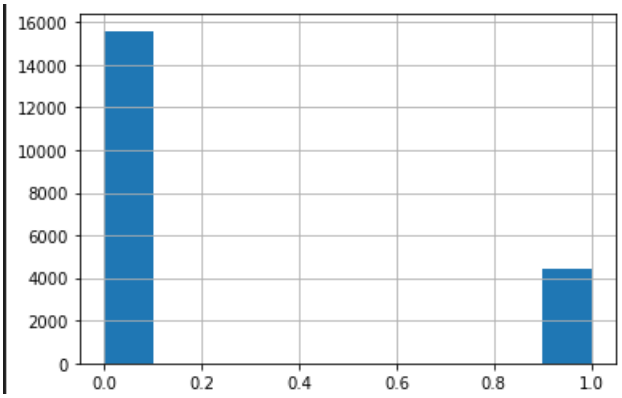
By training and testing our classification models multiple times on different sub-samples of the training dataset, we can have a more accurate overview of how well each model can perform on a data it has not seen. In the **K-fold** cross validation, I will measure the **AUC** and the **F-1 score** of each model after every iteration, and then compute the average of all the scores per model.
After applying a **20-fold** cross validation on the train dataset, the resulting comparison was like this:

	Logistic Regression	LDA	KNN	Decision Tree	SVM
AUC Score	0.7124	0.7111	0.6958	0.6081	0.7129
Accuracy	0.8073	0.8096	0.7917	0.7221	0.7966

	1st Fold	2nd Fold	3rd Fold	4th Fold	5th Fold	6th Fold	7th Fold	8th Fold	9th Fold	10th Fold	11st Fold	12nd Fold	13rd Fold	14th Fold	15th Fold	16th Fold	17th Fold	18th Fold	19th Fold	20th Fold
Logistic Regression	0.7118	0.6969	0.6972	0.7141	0.7030	0.7763	0.6873	0.6836	0.7047	0.7229	0.7090	0.6947	0.7333	0.6927	0.6967	0.6845	0.6986	0.6985	0.7186	0.6852
LDA	0.7116	0.6949	0.7044	0.7143	0.7083	0.7874	0.6881	0.6828	0.7053	0.7228	0.7088	0.6986	0.7367	0.6966	0.6985	0.6864	0.6969	0.6978	0.7236	0.6847
KNN	0.6876	0.6769	0.6626	0.7131	0.7303	0.7499	0.6851	0.6582	0.6870	0.7059	0.6883	0.7232	0.7002	0.6925	0.6718	0.6612	0.6584	0.6859	0.7410	0.6629
Decision Tree	0.6351	0.6461	0.6275	0.6379	0.6392	0.6482	0.5924	0.5931	0.6092	0.6078	0.5912	0.5865	0.6224	0.6180	0.6017	0.5874	0.6044	0.6172	0.6307	0.6208
SVM	0.7149	0.6967	0.7097	0.7178	0.7115	0.7915	0.6894	0.6831	0.7076	0.7235	0.7058	0.7013	0.7386	0.6979	0.6972	0.6855	0.6993	0.6994	0.7229	0.6877

Using this 20-fold cross validation, the *SVM* outperforms the four other classification models by an **AUC score** of 0.7129 and **F-1 score** of 0.7966. As for the second table, we can see how the accuracy score varies from one fold to another, what emphasizes the importance of cross validation over relying just on one simple training and validation sets.

We can also notice that the AUC scores are low, and when digging into the distribution of the target variable we can see that the principle reason behind this is the **imbalance** of our dataset as you can see from the histogram below:



In order to fix this issue, I opted to first oversampling the default class using **SMOTE** with a ratio of **0.6**, and I ended up having a total number of **14490** 0s and **8694** 1s.

After applying this technique and re-doing cross validation on the train set, we can remark as shown below, an increase in the AUC score for all the models:

	Logistic Regression	LDA	KNN	Decision Tree	SVM
AUC Score	0.7169	0.7164	0.8029	0.7084	0.7178
Accuracy	0.7337	0.7338	0.7469	0.7221	0.7340

3. Hyperparameter Tuning

For this task of choosing the optimal hyperparameters for each model, I chose to use one of the most popular approaches of tuning which is **RandomizedSearchCV()**.
In Randomised Grid Search Cross-Validation, fisrt thing to do, is to create a grid of hyperparameters that need to be optimised with a bunch of values that we want to try out, using a dictionary with the key-value pairs as shown below for the decision tree classifier:

```
# define search space
space = dict()
space['criterion'] = ["gini","entropy"]
space['splitter']=["best", "random"]
space['min_samples_leaf']=[1,2,3,4,5,6,7,8,9,10]
space['min_weight_fraction_leaf']=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
space['max_features']=["auto","log2","sqrt",None]
space['max_leaf_nodes']=[None,10,20,30,40,50,60,70,80,90]
```

This method uses cross validation to evaluate model's performance, but instead of trying all possible combinations of hyperparameters, it randomly chooses a values for each hyperparameter from the dictionary and evaluates the model using that combination.

After running the algorithm over 20 folds cross validation, we can get the best combination based on **F1 score**, since it is a better scoring metric than the **accuracy** when dealing with imbalanced data, as follows:

```
Best F1: 0.6292657437386413
Best Hyperparameters: {'p': 1, 'n_neighbors': 8}
```

Once the Randomised Grid Search Cross-Validation is applied on all the 5 classification models, we obtain the optimal hyperparameters (in terms of cross-validation), and we can move to evaluate these models.

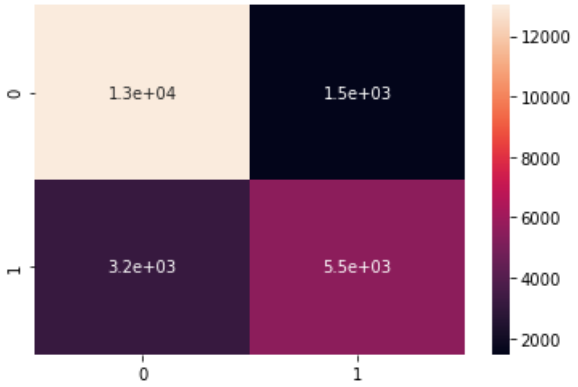
4. Model evaluation

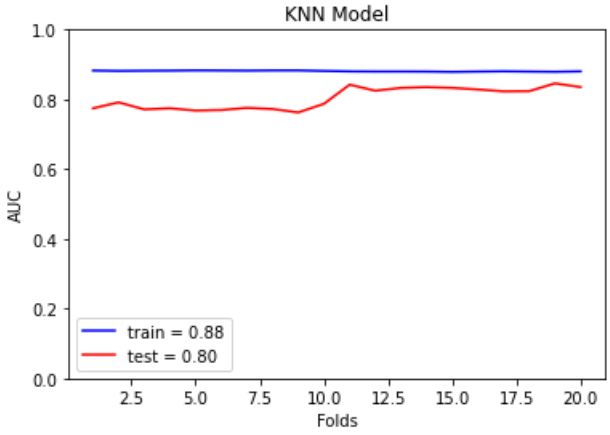
In order to score our models' performance and do the comparison, I used cross validation over the models again using the optimal hyperparameters, and I calculated the F1 score, and the AUC.
From the table of measures below, we can notice that the best model is the **KNN** one with an AUC of **0.80** and a F-1 score of **0.63** for the test and **0.85** AUC and **0.70** for the F-1 score for the train.

	Logisitc Regression	LDA	KNN	Decision Tree	SVM
Test AUC Score	0.719959	0.712576	0.803573	0.681477	0.719362
Test F-1 score	0.550787	0.544861	0.628151	0.461304	0.592504
Train AUC Score	0.720755	0.713417	0.881195	0.686848	0.720220
Train F-1 score	0.550548	0.546177	0.702950	0.460256	0.593035

5. Debugging

After choosing the best model to use for classification for this dataset, I plot both the **confusion matrix**, and the **K-fold curve** to see whether the model is learning for both 0's and 1's and the difference between the AUCs for train and test sets.





As you can see above, the model is capturing a reasonable number of 1s and 0s and the difference between the AUc for test and train sets over the cross validation folds is around 0.05 which we can tolerate, and I assume that the model is not overfitting.

References

- https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/
- https://www.knowledgehut.com/blog/data-science/linear-discriminant-analysis-for-machine-learning
- https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/
- https://www.sciencedirect.com/topics/computer-science/linear-discriminant-analysis
- https://towardsdatascience.com/linear-discriminant-analysis-explained-f88be6c1e00b
- https://www.sciencedirect.com/topics/medicine-and-dentistry/logistic-regression-analysis
- https://towardsdatascience.com/logistic-regression-explained-9ee73cede081
- https://medium.com/data-science-group-iitr/logistic-regression-simplified-9b4efe801389#:~:text=The%20idea%20of%20Logistic%20Regression,two%20values%2C%20pass%20and%20fail.
- https://medium.com/analytics-vidhya/what-is-the-logistic-regression-algorithm-and-how-does-it-work-92f7394ce761
- https://www.saedsayad.com/k_nearest_neighbors.htm#:~:text=K%20nearest%20neighbors%20is%20a,as%20a%20non%2Dparametric%20technique.
- https://medium.com/analytics-vidhya/pros-and-cons-of-popular-supervised-learning-algorithms-d5b3b75d9218
- https://www.kdnuggets.com/2020/11/most-popular-distance-metrics-knn.html#:~:text=The%20Hamming%20distance%20method%20looks,between%20the%20two%20binary%20vectors.
- https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761
- https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4
- https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html#:~:text=The%20goal%20of%20using%20a,prior%20data(training%20data).&text=In%20Decision%20Trees%2C%20for%20predicting,the%20root%20of%20the%20tree.
- https://medium.com/swlh/decision-tree-classification-de64fc4d5aac
- https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052
- https://analyticsindiamag.com/7-types-classification-algorithms/#:~:text=Classification%20model%3A%20A%20classification%20model,of%20a%20phenomenon%20being%20observed.
- https://scikit-learn.org/stable/modules/svm.html
- https://www.youtube.com/watch?v=efR1C6CvhmE
- https://www.analyticsvidhya.com/blog/2020/10/the-mathematics-behind-svm/
- https://www.researchgate.net/publication/301876055_Multilevel_Weighted_Support_Vector_Machine_for_Classification_on_Healthcare_Data_with_Missing_Values