Predict the survival rate on the Titanic

Project group 55 | Filip Muntean (2663515), Sven Lankester (2668125), Andrej Tibenský (2672988), Damla Ural (2672246), Victor Gavrilovici (2670026)

Vrije Universiteit Amsterdam - Faculty of Science
Computer Science Department
Machine Learning (X_4001554)

March 2022

Abstract:

The aim of this report is to predict the survival rate on the Titanic based on the ship's attendees. The data was collected from a project competition on Kaggle.com, where we analyzed almost *900 instances* of our features, from the competition called *"Titanic - Machine Learning from Disaster"*. It was separated into a training set and a test set by the creators of the competition. During this research, various models were trained, including Naive Bayes, KNN, Random Forest, and Neural Networks, in order to achieve the best results for our tests.

1 Introduction

In this research project, many algorithms were tested in order to predict with the least possible recall, the survival rate on the Titanic. The main research question that this research paper focuses on is the following: *"Which machine learning model will most accurately predict whether someone survives the Titanic or not?"*. Our dataset contains features such as name, age, sex, and other characteristics, as well as embarking details. The models we used were, in some part, chosen based on the type of data that the dataset contains, and evaluating the pros and cons, whether the model performs better or worse, etc. Among four models, those which we deemed the most suitable were chosen for our dataset. As studies have shown, Naive Bayes performs relatively well on large and complex datasets [1]. We tried to alternate our results, by selecting other models such as Random Forest, performing rather well on datasets containing numerical and categorical data. Our Neural Network model is expected to perform best, as it's the most complex.

2 Data Inspection

2.1 Data inspection

The first step was to inspect the data. Particularly, the presence of null values and outliers were observed along with the possible important features. Consequently, null values were found both in the train data and test data. The features that were missing data were the *age* and *embarking details*. The embarking information could possibly give us an idea about the location of a passenger during the incident, however since it is not possible to know if the passenger was in the cabin or not it is not of great importance. Age, however, is quite an important feature that affects the possibility of survival and therefore cannot be eliminated as a whole column.

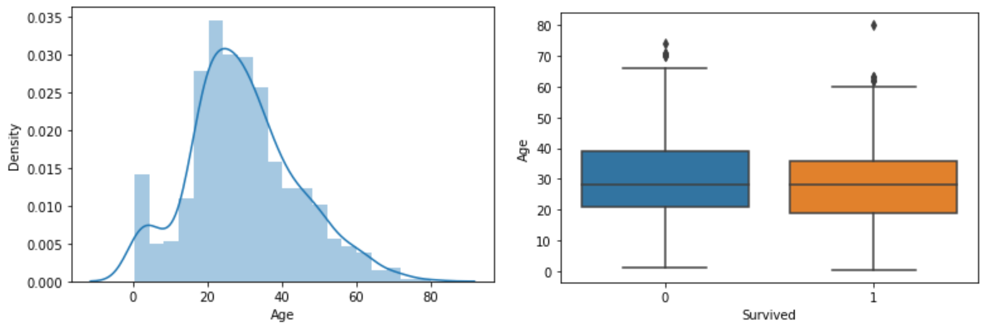

Figure 1: The age distribution and its relation to survival rate

In order to support this claim, it is useful to take a look at this feature a little closer. The distribution of the age looks a bit right-skewed with a couple of natural outliers. However, it looks mostly like a normal distribution. It is therefore possible to leave the outliers in the dataset.

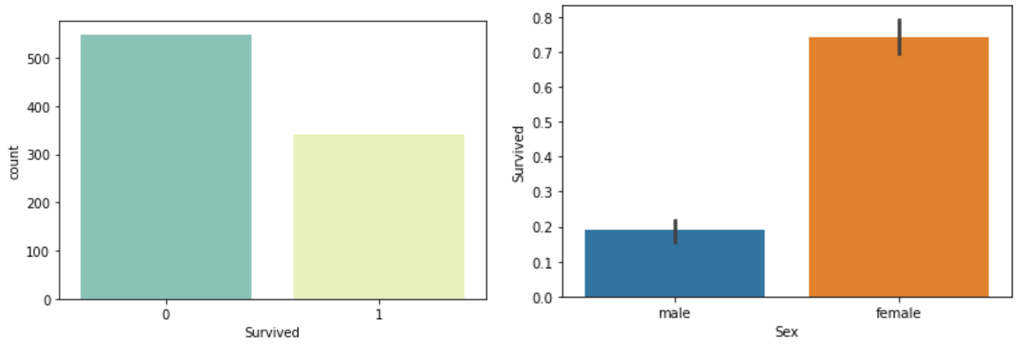

Figure 2: The survival rate of the overall population and of different sexes

Another critical feature that was observed was the sex of passengers. The male survival rate was crucially less compared to the female survival rate on the ship. Approximately, 74% of the female population has survived compared to only 18% of the male population. As for the overall population, we can observe that most people ended up not surviving with a survival rate of 38%.

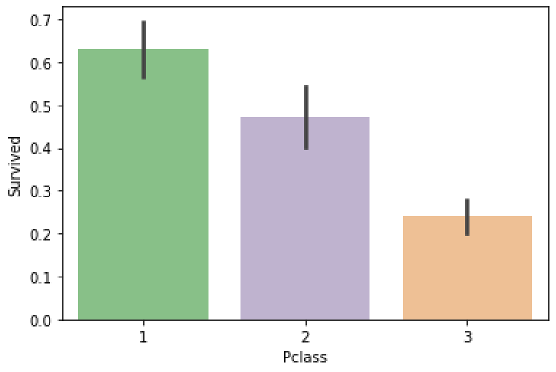

Figure 3: The survival rate of different classes

A useful feature was also the passenger class. This feature describes the *class* of the ticket for each passenger with 1 being first-class, 2 being second-class, and 3 being third-class. As we can observe from

Figure 2, most people who have survived were from the first class which gives us a clear idea about the profile of a possible survivor.

2.2 Data preparation

We tested four models, in our attempt to find out which one would perform best, taking into account our dataset. The Naive Bayes classifier is most suitable for the classification of discrete features that are categorically distributed, while KNN is more so universal since it can be used for any type of data, whether that is continuous, discrete, ordinal or categorical. The Random Forest model, on the other hand, uses sorting/categorization techniques to convert continuous values into discrete ones[2]. When manipulating the data with the Neural Network model, it requires the data to be numerical, thus, we had to encode it, before it could fit and evaluate a model. In order to achieve the latter, we used a Label Encoder from Sklearn, which encodes target labels with a value between 0 and n_classes-1. Next, we split the 'cabin' label into a 'deck' which is the floor of the cabin and 'room' which is the room number on the deck.

3 Baseline

With the prepared data, we select a baseline to compare our future models against to see if there is a significant enough improvement to make it worth using the models. Due to the notable class imbalance of survivors compared to deaths, where death is the case for 62% of the passengers in the data and the leftover 38% are the survivors, we can choose our baseline to be a model which always picks the majority class, in which case its accuracy will be 62%. Our goal with the models we try out is to get an accuracy well above 62% to show there is a correlation between the features and an individual's survival rate. This baseline will be the bare minimum we want to reach, however, we saw one very basic correlation in the data that we would like to use as our target accuracy to surpass. If you have a model that predicts every woman to survive and every man to die, you get an accuracy of 78%. This is a reasonably accurate model already given the fact that deaths are something very hard to predict, definitely since it also corresponds to features unknown to us, such as the location of the passenger at the time of the sinking. For this reason, we want to at least reach an accuracy above 62% for our models but aim to go beyond 78% accuracy.

4 Methodology

One of the most important tasks in model training, is experimenting with different hyperparameters [3]. For further training we have decided to pick four feature sets that we will use to train and test the models and see how well they classify the data. Our original dataset has 10 features that we could use to train the models and 1 target. We also split the Cabin feature into a deck and room number for an extra feature. The rest of this section describes in detail the algorithms that we have used.

4.1 Naive Bayes

The Naive Bayes method is a set of unsupervised algorithms based on Bayes' theorem with the naive assumption of conditional independence between every pair of features given the value of the class variable[4]. The naive assumption represents independence among the features and is expressed by the following formula:

$$P(y|x_1,\ldots,x_n) = \frac{P(x_1|y)P(x_2|y)\ldots P(x_n|y)P(y)}{P(x_1)P(x_2)\ldots P(x_n)}$$

Using the naive conditional independence assumption that:

$$P(x_i|y, x_1,\ldots, x_{i-1}, x_{i+1},\ldots, x_n) = P(x_i|y)$$

For all i, this relationship is simplified to:

$$P(y|x_1,\ldots,x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1,\ldots,x_n)}$$

Naive Bayes provides a mechanism for using the information in sample data to estimate the posterior probability *P(y | x)* of each class y given an object x. Finally, the Naive Bayes classifier is expressed by the following formula:

$$\hat{y} = argmax_{k\in\{1,\ldots,K\}} p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$$

The Naive Bayes has low variance, since it does not utilize search but at the cost of high bias. This algorithm contains, among others, computational efficiency, since training time is linear, with respect to the training examples and the number of attributes. Moreover, classification time is also linear, with respect to the number of attributes and unaffected by the number of examples. The algorithm always uses all attributes for all predictions and is therefore somewhat insensitive to noise. It is also insensitive to missing values, meaning that, since it uses all attributes for all predictions, even if an attribute value is missing, information is still gathered from the others. The Naive Bayes classifier is quite fast compared to other methods. In the case of continuous data, there need to be some assumptions regarding the distribution of values of each feature. This type of classifier has limited options for parameter tuning like *alpha = 1* for smoothing, *fit_prior=[True|False]* to learn class prior probabilities or not. It applies to our model since it is easy and fast to predict the class of a test data set, while also performing well in multi-class prediction.

4.2 KNN

The K-Nearest Neighbors classifier can consider exactly one nearest neighbor, which is the closest training data point to the point we want to make a prediction for. Sometimes, instead of considering only the closest neighbor, we can also consider an arbitrary number of k neighbors. Using a single neighbor results in a decision boundary that follows the training data closely, however, considering more neighbors leads to a smoother decision boundary. Of course, when considering more neighbors, the accuracy of the model drops, and the model becomes simpler. The algorithm is mainly based on the Euclidean distance between a test sample and the specified training sample[5]. This distance is defined as follows:

$$d(x_i, x_l) = \sqrt{(x_{i1} - x_{l1})^2 + (x_{i2} - x_{l2})^2 + \ldots + (x_{ip} - x_{lp})^2}$$

Increased performance is sometimes achieved through feature transformation, which is usually done prior to classification. The two main feature transformations that are used are standardization and fuzzification.

On one side, standardization removes scale effects by features with different measurement scales, while fuzzification uses uncertainty in feature values in order to increase the classification performance. Finally, the usual way in which to check prediction is cross-validation[6]. This process implies the resampling procedure and uses a parameter k, which correlates to the number of groups that the dataset is split into. We chose to use KNN in our project since our dataset is relatively small and we deemed that the algorithm would be suitable since it works well with small datasets. It is fast, and easy to implement, even though it is sensitive to noisy data, missing values, and outliers. We handled the latter during the Data Inspection (see section 2.1).

Choosing hyperparameter k

The most important part of making a KNN classifier is choosing the right hyperparameter. We use the mean of cross-validation scores with 10 splits. This is both because our test set is too small to split into a test and validation set and because it helps with overfitting by testing the hyperparameter on multiple validation sets. Figure 3 shows that the smaller feature sets first scale linearly before leveling off. Meanwhile the larger feature sets start off at a lower value before surpassing the smaller sets followed by leveling off. Therefore, choosing a k above 20 should give us a better accuracy for the smaller feature sets, while choosing a k around 10 should give us the best consistent accuracy for the larger feature sets and overall. Therefore we will be using feature set 3 and a k of 10.

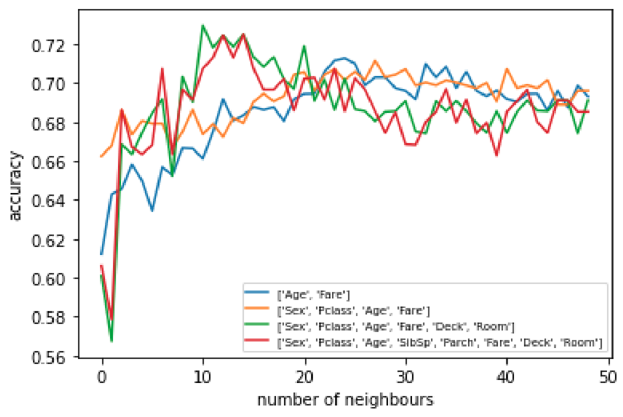

Figure 4: Accuracy of KNN tested on 4 feature sets

4.3 Random Forest

The decision tree classifier is an extremely useful machine learning model since it is easy to implement and performs quite well even on complicated datasets. However, it is very easy to overfit this model as we increase the depth of the tree and the model suffers from high variance. A random forest can be described as an ensemble of randomized decision trees. It uses this disadvantage of easily overfitting and applies *bagging,* which is an ensemble of parallel estimators that each overfit the data and then averages it out which eventually reduces variance [7]. The model simply contains a number of bagged and uncorrelated decision trees that each predicts a class as it normally would and considers the majority decision. The bagging algorithm, also known as bootstrap aggregation works as follows[8]:

$$\widehat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B}\widehat{f}^{b}(x) \text{ } where\ B\ is\ the\ number\ of\ different\ training\ sets$$

The random forest classifier can easily be used with Python and Sklearn. There are naturally a couple of hyperparameters to consider, some that we have already mentioned before, and the most crucial ones are stated below:
n_estimators: the number of trees in the forest, max_depth: the maximum depth of the tree, random_state: controls both the randomness of the bootstrapping of the samples used when building trees and the sampling of the features to consider when looking for the best split at each node

Choosing hyperparameter max depth

We once again use the mean of 10 cross-validation scores because our test set is too small and because it helps with overfitting. Figure 5 shows that the different feature sets have large differences in accuracy for Random forest with feature set 1 having the lowest accuracy and feature set 2 having the highest. The different depths don't influence the accuracy beyond 3 except for a consistent spike at 8 for feature set 2. Therefore we will be using depth of 8 with feature set 2.

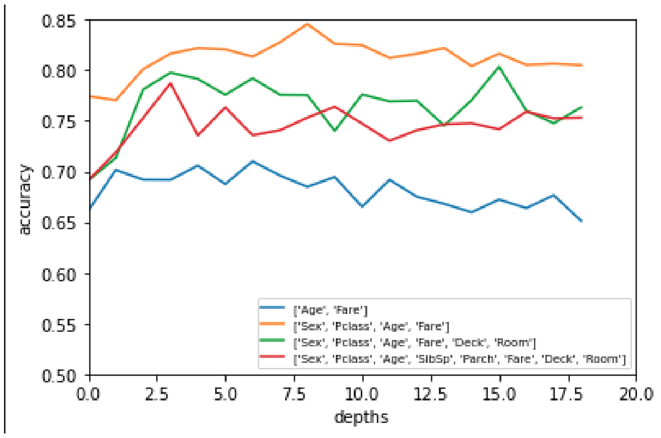

Figure 5: Accuracy of Random forest on 4 feature sets

4.4 Neural Network

Artificial neural networks are an attempt to mimic how the human brain learns, although it's not complete replication, the concept of an artificial neural network was inspired by the working of biological neurons where based on the input, certain neurons fire or "activate". An artificial neural network usually consists of 3 main components:

1. The input layer: This is the layer where your neural network takes in the features as a tensor, so they can be fed forward into the network where the weight of each feature decides its impact on the output.
2. The hidden layer(s): This is where the actual processing of the information is mostly done. The neural network takes features from the input and by applying weights to them turns combinations of inputs into new features. This component should be wider than the input layer.
3. The output layer: This is where the output of the hidden layer is used as input for the eventual result the neural network produces. This layer is often a distribution of probabilities that the neural network thinks the input corresponds to.

We construct the neural network using Keras where we pre-process the data using Sklearn to turn categorical data into numerical data for our network to process. After the data was pre-processed as discussed in section 2.2, we apply standardization to it[9], in order to normalize our data. This was shown to bring a significant performance increase to our network, as without standardization our accuracy would consistently be below 80%. We are left with around 750 instances in our training set, so we use stratified k-folds cross-validation to use as much as possible of the data in training while keeping the classes balanced in each fold. Therefore all mentions of accuracy on our hyperparameter selection stage include both the mean of the accuracy obtained on all folds and its standard deviation. As an activation function we use the "ReLU" function on all layers except the output, which uses the sigmoid function in order to produce a probability output in the range of 0 to 1. The "ReLU" function was used as it performed slightly better compared to the sigmoid in every testing configuration. We have used binary cross-entropy as our loss function due to it being well suited for our binary classification task.

The first hyperparameter tuning was done for the number of neurons used in a layer, which was done on the input layer, with no hidden layers.

Number of neurons	128	64	32	16	8
Validation accuracy mean (standard dev.)	81.67% (5.41%)	80.81% (3.56%)	80.96% (3.65%)	81.08% (3.56%)	80.81% (3.76%)

Table 1: Validation accuracy mean with different number of neurons

We decided to continue with 16 neurons per layer, as it showed similar performance to the 128 neuron per layer layout, and it had a lower standard deviation, as well as requiring less computation, which was relevant for larger networks.
We also experimented with the number of hidden layers to see if we can get more accurate results:

Number of hidden layers	None	1	2	3
Validation accuracy mean (standard dev.	81.08% (3.56%)	81.36% (3.66%)	82.06% (5.69%)	81.38% (3.43%)

Table 2: Validation accuracy mean with different number of hidden layers

We use the ADAM optimizer[10] where we experimented on a network with 2 hidden layers with 16 and 4 with the following learning rates:

ADAM learning rate	0.001	0.01	0.0005	0.00025	0.0001
Validation accuracy mean (standard dev.)	81.79% (3.77%)	77.58% (7.07%)	81.79% (4.01%)	81.80% (3.51%)	80.52% (5.35%)

Table 3: Validation accuracy mean with different numbers of ADAM learning rate

The final two hyperparameters to choose were number of epochs and batch size.

Number of epochs	25	50	100	150
Validation accuracy mean (standard dev.)	81.37% (2.30%)	81.37% (2.84%)	80.95% (5.18%)	80.11% (2.75%)

Table 4: Validation accuracy mean based on different epochs

So we combined these results and decided on the following hyperparameters:

Number of neurons per hidden layer	16
Amount of hidden layers	2
ADAM Learning rate	0.00025
Number of epochs	100
Batch size	5

Table 5: Final hyperparameters for the neural network

5 Results

In this section, the feature sets that were described before will be mentioned as follows:

Set 1: Age, Fare

Set 2: Sex, Pclass, Age, Fare

Set 3: Sex, Pclass, Age, Fare, Story, Room

Set 4: Sex, Pclass, Age, SibSp, Parch, Fare, Story, Room

5.1 Naive Bayes

We used multiple feature sets on the naive Bayes model; the results can be seen below. There is very little variance in the accuracy between each feature set which could follow from the simple nature of the naive Bayes model, as it assumes independence between the features.

Feature set	Set 1	Set 2	Set 3	Set 4
Accuracy	0.81	0.74	0.81	0.81

Table 6: Accuracy of training the model on each of the feature sets

5.2 KNN

The accuracy for feature set 3 and a k of 10 is 0.63. When testing on other feature sets, the accuracy stayed pretty consistent. The accuracy is lower because we have corrected for overfitting. Also we can see the detection error tradeoff from the DET curve.

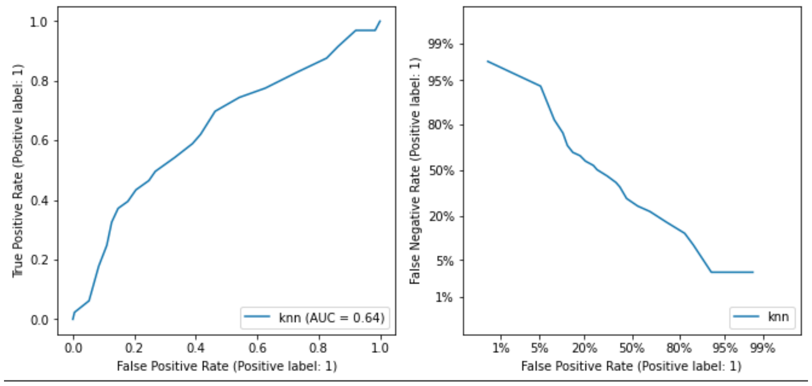

Figure 6: ROC curve (left) DET curve (right)

5.3 Random Forest

The random forest models that we have trained performed quite well in general. The accuracy for feature set 2 and depth 8 was 80.25% however the other feature sets performed similarly well with accuracies of 75 - 80%. We can see this from the ROC curve in Figure 7 where the TPR grows very fast and the AUC is the largest of the 3 models.

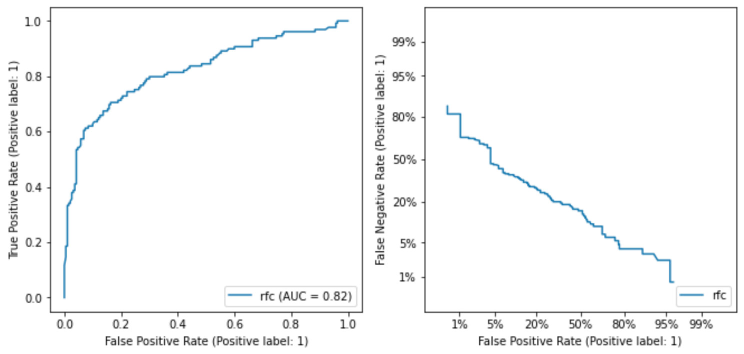

Figure 7: ROC curve (left) DET curve (right)

5.4 Neural Network

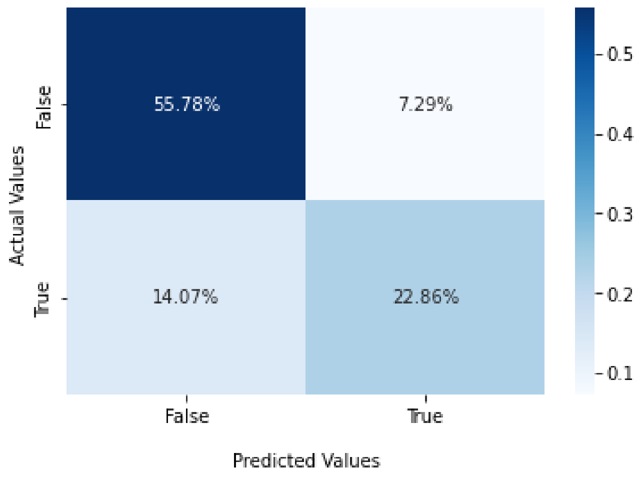

Figure 8: Neural Network results confusion matrix

The NN has an accuracy of 78% on our test set, and based on the confusion matrix we can deduce that it prefers negative classification, which should be expected as there is such a class imbalance in all of our data, in the form of more deaths.

6 Discussion:

The model that had the best accuracy on the test set in predicting whether or not someone would survive the Titanic with the hyperparameters we found was the naive Bayes classifier. The summary of our findings can be found in the table below:

Model	**Accuracy**
Naive Bayes	80.72%
KNN	63.85%
Random forest classifier	80.25%
Neural network	78.64%

In this project we definitely faced some difficulties with the somewhat small dataset not giving us results far above the only-females-survive model. Another cause of this is that someone surviving the Titanic can be more or less random, as someone with the perfect specifications could be in the room where the Titanic was hit and instantly die. However, this does not mean that the accuracy of our models cannot be improved any further. There are many extra experiments that could be run in future research to see whether or not accuracy could be improved. Examples of this would be deriving several new features from the original ones, adjusting the hyperparameters for the complex models such as a neural network even more thoroughly or of course trying other machine learning models that could perform well on this problem.

References:

[1] Ekin Ekinci, Sevinç İlhan Omurca, Neytullah Acun. A Comparative Study on Machine Learning Techniques using Titanic Dataset, 7th International Conference on Advanced Technologies (ICAT'18), May 2018

https://www.researchgate.net/profile/Neytullah-Acun/publication/324909545_A_Comparative_Study_on_Machine_Learning_Techniques_Using_Titanic_Dataset/links/607533bc299bf1f56d51db20/A-Comparative-Study-on-Machine-Learning-Techniques-Using-Titanic-Dataset.pdf

[2] Gérard Biau, Erwan Scornet, A random forest guided tour, April 2016, https://link.springer.com/article/10.1007/s11749-016-0481-7

[3] Xingyou Song, Yilun Du, Jacob Jackson, An empirical study on Hyperparameters and their Interdependence for RL Learning, 2019, https://arxiv.org/pdf/1906.00431.pdf

[4] Geoffry I Webb, Monash University, Encyclopedia of Machine Learning and Data Mining, Springer, 2017, https://www.researchgate.net/profile/Geoffrey-Webb/publication/306313918_Naive_Bayes/links/5cab15724585157bd32a75b6/Naive-Bayes.pdf

[5] Leif E. Peterson, K-nearest neighbor, Scholarpedia, 4(2):1883, 2009, http://scholarpedia.org/article/K-nearest_neighbor

[6] Payam Rafaelizadeh, Lei Tang, Huan Liu. Cross-Validation, 2016.

[7] VanderPlas, Jake. *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media, Incorporated, 2016.

[8] Hastie, Trevor, et al. *An Introduction to Statistical Learning: With Applications in R*. Springer US, 2021.

[9] Jason Brownlee, Develop Deep Learning Models on Theano and TensorFlow Using Keras, 2020, https://books.google.nl/books?hl=en&lr=&id=K-ipDwAAQBAJ&oi=fnd&pg=PP1&dq=standardization+using+keras&ots=oqTv1J_lwn&sig=LMFdtllTaKZl248aUJ1GcwoqJbY&redir_esc=y#v=onepage&q=standardization%20using%20keras&f=false

[10] Diederik P. Kingma, Jimmy Ba. *Adam: A Method for Stochastic Optimization*, 2015.