Prediction of Convergence rate for a Bank Marketing Dataset using Machine Learning

Anuja Nagare , Raunak Dey, Shubha Mishra, Zahra (Parya) Jandaghi
University of Georgia

***Abstract* - In this project, various machine learning algorithms are used in order to predict convergence rate of customers subscribing to a term deposit, given data about direct marketing campaigns (phone calls) of Portuguese banking institute. Prediction accuracies of these algorithms are compared with each other and it is observed that Neural Networks performs better than other supervised algorithms.**

Index Terms – Machine Learning, Bank Marketing dataset, prediction.

1. INTRODUCTION

Machine learning is used in various applications to investigate ways to learn from data and make predictions on data[1]. Our goal is classification of data into two classes, in order to predict convergence rate of clients subscribing to the term deposit. We are comparing various supervised learning algorithms to find out, which machine learning technique provides optimal accuracy for prediction, based on the dataset.

Figure 1. show the problem definition of the project.

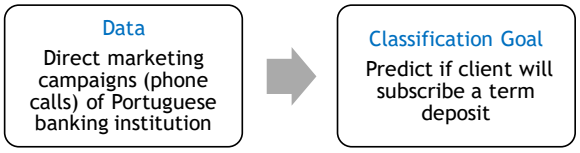

Figure 1. Problem definition

2. DATASET

The data is related with direct marketing campaigns based on phone calls, by a Portuguese banking institution. Dataset has 362451 number of hits, last updated date - 05/05/2017. This dataset is multivariate, with 45211 number of instances. After balancing the positive and negative instances, 9280 instances are extracted.

There are total 21attributes including the target attribute, which shows that the client will subscribe a term deposit. Figure 2. Shows all the attributes in the dataset.

Figure 2. Bank Marketing Dataset

Attribution information has 4 types of input variables which can be given as follows:

bank client data:

1) age (numeric)
2) job : type of job (categorical: 'admin.', 'bluecollar', 'entrepreneur', 'housemaid', 'management', 'retired', 'self-employed', 'services', 'student', 'technician', 'unemployed', 'unknown')
3) marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)
4) education (categorical: 'basic.4y', 'basic.6y', 'basic.9y', 'high.school', 'illiterate', 'professional.course', 'university.degree', 'unknown')
5) default: has credit in default? (categorical: 'no','yes','unknown')
6) housing: has housing loan? (categorical: 'no','yes','unknown')
7) loan: has personal loan? (categorical: 'no','yes','unknown')

related with the last contact of the current campaign:

8) contact: contact communication type (categorical: 'cellular','telephone')
9) month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
10) day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11) duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be

included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

other attributes:

12) campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13) pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14) previous: number of contacts performed before this campaign and for this client (numeric)
15) poutcome: outcome of the previous marketing campaign (categorical: 'failure', 'nonexistent', 'success')

social and economic context attributes

16) emp.var.rate: employment variation rate - quarterly indicator (numeric)
17) cons.price.idx: consumer price index - monthly indicator (numeric)
18) cons.conf.idx: consumer confidence index - monthly indicator (numeric)
19) euribor3m: euribor 3 month rate - daily indicator (numeric)
20) nr.employed: number of employees - quarterly indicator (numeric)

The output variable/ desired target is the last attribute y which has, the client information in binary format (i.e. 'yes','no') and tell if client subscribed to a term deposit.

3. MACHINE LEARNING TECHNIQUES USED FOR CLASSIFICATION

3.1. k-nearest neighbors

The k-nearest neighbors algorithm (k-NN) is a non-parametric classification technique[1]. Input consists of the k closest training examples in the feature space and output depends on a class membership.

A client is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor[2].

sklearn library makes use of following parameters for knn:
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

The accuracy obtained with KNN is 0.859459459459

3.2. SVM Linear

Support Vector Machine is a supervised classification model.

In sklearn library, Linear Support Vector Classification is similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.
This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme[2].

sklearn library makes use of following parameters for svm:
class sklearn.svm.LinearSVC(*penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000*)

The accuracy obtained with SVM Linear is 0.838918918919

3.3. SVM RBF

Sklearn implementation of SVM RBF is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples[2].

The multiclass support is handled according to a one-vs-one scheme. For details on the precise mathematical formulation of the provided kernel functions and how gamma, coef0 and degree affect each other, see the corresponding section in the narrative documentation: Kernel functions.

sklearn library makes use of following parameters for Radial Basis Function:
class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None

The accuracy obtained with SVM RBF is 0.784864864865

3.4. Quadratic Discriminant Analysis

A classifier with a quadratic decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule. The model fits a Gaussian density to each class in sklearn libraby, it uses following parameters for QDA[2]:
QuadraticDiscriminantAnalysis(priors=None, reg_param=0.0, store_covariances=False, tol=0.0001)[source]

The accuracy obtained with QDA is 0.795675675676.

3.5. Logistic regression

Logistic regression was developed by statistician David Cox in 1958. It is one the most-widely used machine learning algorithms used for solving regression as well as classification problems.

Some of the major research areas which use this algorithm includes machine learning, almost all medical fields, business, marketing and social sciences. For example, the Trauma and Injury Severity Score (TRISS), which is widely used to predict mortality in injured patients, was originally developed by Boyd et al. using logistic regression.[4] It may be used to predict whether a patient has a given disease (e.g. diabetes; coronary heart disease), based on observed characteristics of the patient (age, sex, body mass index, results of various blood tests, etc.).

Implementation

Logistic regression have been used in this work to predict the convergence rate of a client using bank marketing dataset, collected from UCI repository, based on 20 other attributes.

The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). In such cases, the dependent variable can only take two values, "0" and "1", which represent outcomes such as pass/fail, win/lose, survived/not and similar outcomes.

Here, the logistic regression model uses similar binary dependent variables to predict whether a client converged or not after learning user behavior on given attributes.

A logistic function or logistic curve is a common "S" shape (sigmoid curve), with equation:

$$f(x) = L/(1+ e^{-k(x-x_0)})$$

where

- e = the natural logarithm base (also known as Euler's number),
- x0 = the x-value of the sigmoid's midpoint,
- L = the curve's maximum value, and
- k = the steepness of the curve.

For implementation purpose, Scikit learn's linear model package have been used. Some essential parameters are as shown below. Modifying some of these parameters showed variations in the prediction out-comes.

1. penalty : str, 'l1' or 'l2', default: 'l2'

Used to specify the norm used in the penalization. The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties.

2. dual : bool, default: False

Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n_samples > n_features.

3. C : float, default: 1.0

Inverse of regularization strength; must be a positive float. Like in sup-port vector machines, smaller values specify stronger regularization.

4. fit_intercept : bool, default: True

Specifies if a constant (a.k.a. bias or intercept) should be added to the decision function.

5. class_weight : dict or 'balanced', default: None

Weights associated with classes in the form {class label:weight}. If not given, all classes are supposed to have weight one.

6. max_iter : int, default: 100

Useful only for the newton-cg, sag and lbfgs solvers. Maximum number of iterations taken for the solvers to converge.

7. random_state : int seed, RandomState instance, default: None

The seed of the pseudo random number generator to use when shuffling the data. Used only in solvers 'sag' and 'liblinear'.

8. solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag'}, default: 'liblinear'

Algorithm to use in the optimization problem.

- For small datasets, 'liblinear' is a good choice, whereas 'sag' is faster for large ones.
- For multiclass problems, only 'newton-cg', 'sag' and 'lbfgs' handle
- multinomial loss; 'liblinear' is limited to one-versus-rest schemes.

3.6. Decision Tree

A decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called "root" that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an internal or test node. All other nodes are called leaves (also known as terminal or decision nodes). In a decision tree, each internal node splits the instance space into two or more subspaces per a certain discrete function of the input attributes values. In the simplest and most frequent case, each test considers a single attribute, such that the instance space is partitioned per the attribute's value. In the case of numeric attributes, the condition refers to a range. Each leaf is assigned to one class representing the most appropriate target value. Alternatively, the leaf may hold a probability vector indicating the probability of the target attribute having a certain value. Instances are classified by navigating them from the root of the tree down to a leaf, per the outcome of the tests along the path.

Naturally, decision makers prefer less complex decision trees, since they may be considered more comprehensible. Furthermore, per Breiman et al. (1984) the tree complexity has a crucial effect on its accuracy. The tree complexity is explicitly controlled by the stopping criteria used and the

pruning method employed. Usually, the tree complexity is measured by one of the following metrics: the total number of nodes, total number of leaves, tree depth and number of attributes used. Decision tree induction is closely related to rule induction. Each path from the root of a decision tree to one of its leaves can be transformed into a rule simply by conjoining the tests along the path to form the antecedent part, and taking the leaf's class prediction as the class value.
Implementation:
Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. For implementation purpose, Scikit learn's linear model package have been used. Some essential parameters are as shown below. Modifying some of these parameters showed variations in the prediction outcomes.
Parameters:
1. criterion: string, optional (default="gini") : The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
2. splitter: string, optional (default="best") :The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.
3. max_features: int, float, string or None, optional (default=None): The number of features to consider when looking for the best split
4. max_depth : int or None, optional (default=None) : The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
5. max_leaf_nodes : int or None, optional (default=None) :Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.
6. class_weight : dict, list of dicts, "balanced" or None, optional (default=None) : Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.
7. min_impurity_split : float, optional (default=1e-7) : Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise it is a leaf.
8. classes_ : array of shape = [n_classes] or a list of such arrays : The classes labels (single output problem), or a list of arrays of class labels (multi-output problem).
9. feature_importances_ : array of shape = [n_features] : The feature importances. The higher, the more important the feature. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance.
10. max_features_ : int : The inferred value of max_features.
11. n_features_ : int : The number of features when fit is performed.
12. tree_ : Tree object :The underlying Tree object

3.7. Ensemble Learning

Those use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. We have several kinds like:

- Bayes Optimal Classifier
- Bayesian parameter averaging
- Bayesian model combination
- Bucket of models
- Stacking

Here we used boosting which involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models mis-classified. In some cases, boosting has been shown to yield better accuracy than bagging, but it also tends to be more likely to over-fit the training data. By far, the most common implementation of Boosting is Adaboost
AdaBoost, short for "Adaptive Boosting", is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire who won the Gödel Prize in 2003 for their work. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing (e.g., their error rate is smaller than 0.5 for binary classification), the final model can be proven to converge to a strong learner.

Here we gained accuracy of 0.85 by using Adaboost.

3.8. Neural Networks

One of the machine learning algorithm we used was neural networks. Artificial Neural Network (ANN) is an intelligent operator inspired somewhat by the work-ing of the human Brain. Artificial Neural Network tries to model the working of the neurons, in particular the process how they connect to each other and are electrically excitable. A neuron maintains a voltage gradient which enable it to trigger and send signals when its excited. The neurons in Artificial Neural Net-work have a threshold function in order to build up before firing a signal to the next neuron down the chain. Artificial Neural Network has weights associated with the connections between neurons which multiplies the input signal by the weights. Artificial Neural Network are particularly useful for classification and prediction problems.

Layers
This consists of the Input Layer, Hidden Layer(s), and the output layer. The input layer is a set of neurons which accept the inputs. They pass the inputs through the neural network

by multiplying the input with the weight and passing it down the neural network to hidden layers. Hidden layers consist of neurons having activation function (tansig, logsig or sigmoid, linear, relu).

Neurons
These mimic the dendrites of a human brain. They consist of an activation function and fires a signal when stimulated to pass information to the next neuron.
Connections and Adjustable weights
Connections mimic the axon of the human brain. They consist of adjustable weights and multiply the values coming from the input node with the corresponding weight along the path and pass it on to the next neuron it connects to.
Bias neurons
These are neurons which are connected to a bias having a value 1 and it has a weight associated with the connection. The purpose of this is to shift the activation function left or right which helps in changing the characteristics of the activation condition to better predict or classify a problem.
Activation Function
These are the triggers which collect information and once the trigger condition is met, fire a signal according to the characteristics of the function.

Deep Neural Network
A deep learning network is an advanced neural network with multiple layers and includes a large number of neurons per layer. As such there are some changes from the regular neural networks. Mean squared error is replaced with the cross-entropy family of loss functions Sigmoid activation functions are replaced with piecewise linear activation functions, e.g., ReLU.
As the number of connections in the deep network is massive in size, it's some-times not optimal to have all the layers connected. And as such we have the different layer structures Fully Connected Layer/dense: All the neurons are connected to the previous layer.
Dropout Layers: For each step of training keep a neuron active with some probability p; otherwise, set it to zero.
Optimization Algorithms: Deep learning network utilizes more than one but less than all of the training examples i.e. a minibatch stochastic method.
Examples of optimization algorithms include the traditional Stochastic gradient descent which has a constant learning rate and algorithms with adaptive learning rate such as Adagrad, RMS prop, Adam, AdaDelta.
Choosing the correct optimization algorithm depends on the problem at hand and it's based on trial and error methods.

Problems with our dataset
We initially implemented WEKA's MLP as well as MATLAB's NNTool to try and obtain a proper prediction. However the results were an abysmal 40-50% in both the cases. We then moved our implementation to Keras in order to implement a deep dense network. However the result didn't improve no matter how deep we made the network. We even tried for a network with 16 layers with first three layers having 32 neurons, the next 3 having 64, and so on with the 16th layer having 2048 neurons. It seemed that for some reason the network was unable to learn any feature. This we can attribute to the millions of parameters created which may potentially require months of training.

Internal Covariant shift
One of the major problems that a deep network suffers from is the internal covariant shift. Formally, covariate shift is defined as a change in the distribution of a function's domain [Ioffe et al, 2015]. This is majorly due to the all the layers updating their weights at the same time. Thus the dependencies between the net-work layers keep on changing at every stage. This may cause some of the changes to drastically differ from its initial state. Our aim is to tune the parameters slowly however due to Internal Covariant Shift the parameters may change drastically.

Deep network with Batch Normalization
We implemented the [Ioffe et al, 2015] algorithm for Batch Normalization using the Keras Deep Learning Library. The high level idea includes that we normalize the mean and standard deviation of the minibatches and add a parameter to shift it and make it trainable during the training phase of the network. During the inference/evaluation phase the offset created due to this normalization is matched by a modified mean and standard deviation calculation which aims to nullify the effect of the Batch Normalization layers during testing such that the output directly depends only on the input. The network summary is provided in figure 1. We had a total of 2,115,201 parameters with 2,107,265 trainable ones 7,936 non trainable ones.

We obtained a ROC value of 92.8% on the testing set as mentioned in the previous sections.

4. EXPERIMENTS AND RESULTS

It can be seen that Neural Networks performed the best, the prediction accuracy for neural networks is 92.8. Following table shows the prediction accuracy for each machine learning method.
Table 1. Prediction Accuracy for each machine learning algorithm

Sr. No.	Machine Learning Algorithm	Accuracy
1.	SVM RBF	0.784864864865
2.	Quadratic Discriminant Analysis	0.795675675676
3.	SVM Linear	0.838918918919
4.	Decision Trees	0.841081081081
5.	Ensemble Learning	0.85
6.	K-Nearest Neighbours	0.859459459459
7.	Logistic Regression	0.867027027027
8.	Ada-Boost	0.872432432432
9.	Random Forest	0.877837837838
10.	Neural Network	0.928

Figure 3 show the graph for machinea leaning technique vs the prediction accuracy and figure 5. shows the network architeccture of a neural network.

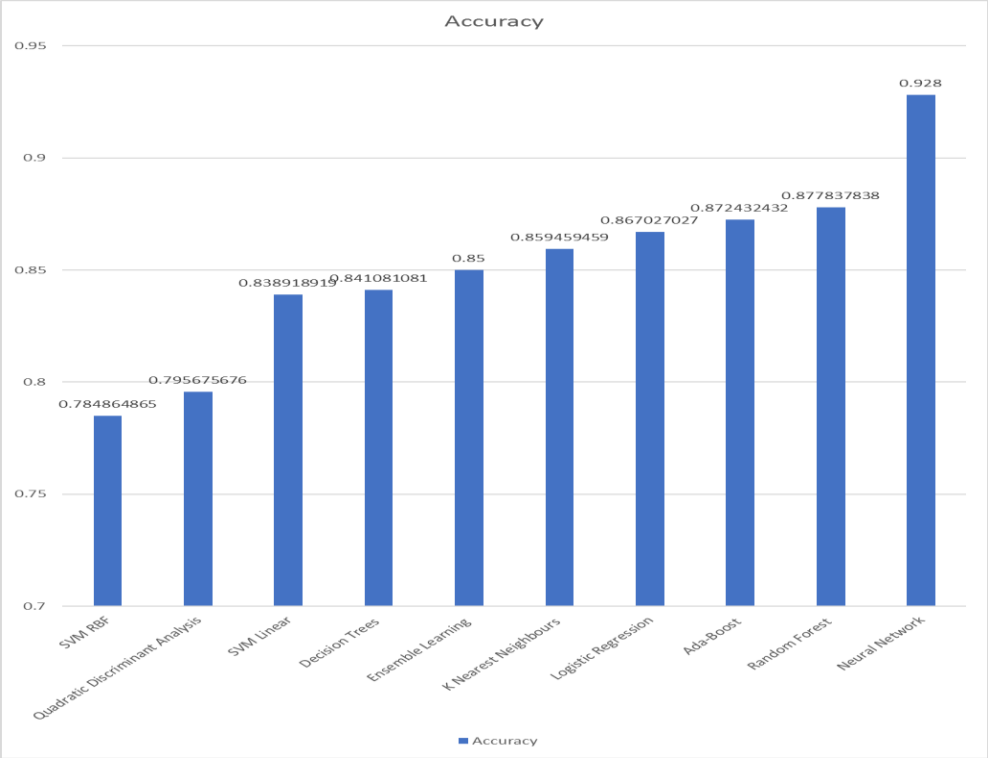

Figure 3: Machine Learning Technique Vs Prediction Accuracy

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 64)	1344
batch_normalization_1 (Batch	(None, 64)	256
dense_2 (Dense)	(None, 64)	4160
batch_normalization_2 (Batch	(None, 64)	256
dense_3 (Dense)	(None, 128)	8320
batch_normalization_3 (Batch	(None, 128)	512
dense_4 (Dense)	(None, 128)	16512
batch_normalization_4 (Batch	(None, 128)	512
dense_5 (Dense)	(None, 256)	33024
batch_normalization_5 (Batch	(None, 256)	1024
dense_6 (Dense)	(None, 256)	65792
batch_normalization_6 (Batch	(None, 256)	1024
dense_7 (Dense)	(None, 512)	131584
batch_normalization_7 (Batch	(None, 512)	2048
dense_8 (Dense)	(None, 512)	262656
batch_normalization_8 (Batch	(None, 512)	2048
dense_9 (Dense)	(None, 1024)	525312
batch_normalization_9 (Batch	(None, 1024)	4096
dense_10 (Dense)	(None, 1024)	1049600
batch_normalization_10 (Batc	(None, 1024)	4096
dense_11 (Dense)	(None, 1)	1025

Figure 4.Network Architecture

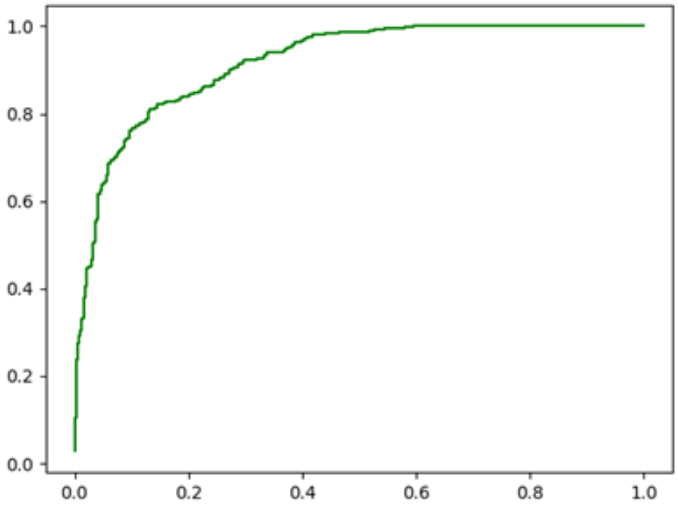

Figure 5. ROC Curve

Figure 5 shows the ROC curve for the NN result.

CONCLUSION

Classification and prediction accuracy for finding the convergence rate of customers for the bank marketing dataset was calculated using various machine learning techniques as mentioned above. The prediction accuracy for Neural Network was observed to be the best.

Also, the prediction accuracy of ensemble learning technique and random forest was observed to be 0.872432432432, 0.877837837838 respectively, by tweaking various parameters while training using different machine learning algorithms we can improve the prediction accuracy further.

REFERENCES

[1] Walker, SH; Duncan, DB (1967). "Estimation of the probability of an event as a function of several independent variables". Biometrika. 54: 167–178. doi:10.2307/2333860.

[2] Jump up^ Cox, DR (1958). "The regression analysis of binary sequences (with discussion)". J Roy Stat Soc B. 20: 215–242. JSTOR 2983890.

[3] Jump up^ Boyd, C. R.; Tolson, M. A.; Copes, W. S. (1987). "Evaluating trauma care: The TRISS method. Trauma Score and the Injury Severity Score". The Journal of trauma. 27 (4): 370–378. doi:10.1097/00005373-198704000-00005. PMID 3106646.

[4] LIBLINEAR – A Library for Large Linear Classification http://www.csie.ntu.edu.tw/~cjlin/liblinear/

[5] Author's Last name, First initial, Middle initial, "Title," *Journal or book (italics),* Vol, No #., date, pp.

[6] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[7] http://scikit-learn.org/stable/supervised_learning.html#supervised-learning

[8] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and Regression Trees", Wadsworth, Belmont, CA, 1984.

[9] T. Hastie, R. Tibshirani and J. Friedman. "Elements of Statistical Learning", Springer, 2009.

[10] Oded Maimon, Lior Rokach, „Data Mining and Knowledge Discovery Handbook", Second Edition, Springer Science+Business Media, p. 149-174

[11] https://archive.ics.uci.edu/ml/datasets/Bank+Marketing