Predicting survival on the Titanic with Machine Learning models

Project Group 79 | Casper Åsblom (2659960)

Abstract

This paper aims to find which of five pre-chosen machine learning algorithms performs best in the task of predicting the survival status of passengers on the Titanic. The algorithms compared are Random Forest, kNN, Stochastic Gradient Descent, Logistic Regression, and Support Vector Machines. After inspecting the data, the methodology of the algorithms are reviewed, and the results of the implementations are finally presented. The algorithm that performed the best in our implementation was the Random Forest algorithm, with an average accuracy of 81.59%. Whether this can be generalized to the task as a whole can however not be concluded.

1 Introduction

In this research project, five different machine learning models will be used to predict the survival of passengers on the Titanic. The data used to train the algorithms is from a Kaggle competition called 'Titanic - Machine Learning from Disaster', and it hold the details of 891 passengers on board and our target variable: whether they survived or not. The objective for each model is to predict the survival of a remaining 418 passengers in a test set where this target variable is unknown. Our goal is ultimately to find which of the five chosen models performs the best at this task, based on their respective accuracy, precision, recall, and F-score.

2 Data inspection and visualization

2.1 Data inspection

The data provided in the competition is two files: train.csv, and test.csv. The train.csv file - our training data set - lists 891 passengers along with 12 features for each. The most notable features are the passengers' name, age, sex, cabin class, number of siblings/spouses aboard, number of parents/children aboard, and our target variable: whether they survived or not (a "1" denoting that the passenger survived, and a "0" denoting that they died). The test.csv file - our testing data set - lists the remaining 418 passengers in the passenger records of the Titanic, along with all the same features except that of survival, which is the value we will attempt to predict.

2.2 Data visualization

In order to find what features will be relevant in our predictions, it is useful to analyze and visualize some aspects of the data. First of all, one could assume that the survival rate of females on board is higher than that of males. When analyzing the "Sex" and "Survived" features in our training data set, we indeed find that whereas approximately 74 percent of the female passengers survived, only about 18 percent of the male passengers did.

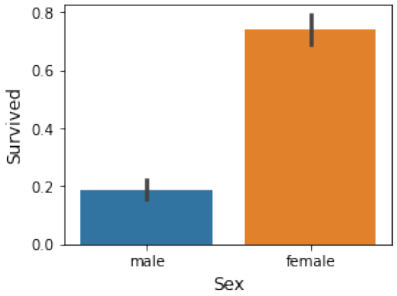

Figure 1: Female vs. male survival rate

Second of all, newborns and children were most likely prioritized on the ship, and their survival rate should as such be higher. Grouping the passengers into age groups, we indeed find that the survival rate is the highest for infants, and steadily declines with the increase of age.

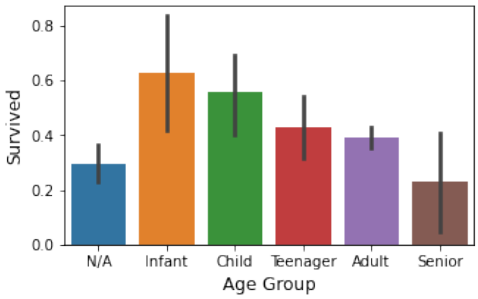

Figure 2: Survival rates by age group

Another feature that likely had great importance is what ticket class the passengers were travelling in. It is rather probable that the passengers who travelled 1st class had a higher survival rate than those who travelled 2nd or 3rd. Drawing a plot, we indeed find a negative linear relationship between survival rate and ticket class. While over 62 percent of the passengers travelling in 1st class survived, this number drops down to 47 percent for 2nd class passengers and 24 percent for those in 3rd class.

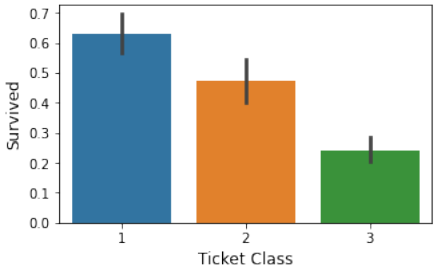

Figure 3: Survival rates by ticket class

Although there are more features we could include from the data set, it could be argued that sex, age, and ticket class are the three most important and independent ones. Fare, for example, should broadly be measuring the same thing as the ticket class, and including it might as such not provide much further insight for the classifiers. In fact, adding more features could potentially do more harm than good, since the more features a model has, the higher is the risk of overfitting. It is therefore advisable to select the most useful features, and discard those whose effects on the final output may be limited. [1]

3 Methods

3.1 Random Forest

Random forest is an ensemble machine learning algorithm whose main concept is to have several decision trees classify a new instance through majority vote. Formally speaking, it is a classifier consisting of a collection of decision trees $\{h(x, \Theta_k), k = 1, ...\}$, where the $\{\Theta_k\}$ are independent identically distributed random vectors, and each tree casts a unit vote for the most popular class at input x. One of the main advantages of random forests is that they always converge - meaning that overfitting is not an issue. [2]

Although there are several hyperparameters of the random forest algorithm which could be tuned to improve performance, the effect of such tuning is rather small compared to other machine learning algorithms. [3] For this reason, the default values specified in scikit-learn will be used in this implementation.

3.2 kNN

kNN, or k-nearest neighbors, is a classification method which takes a positive integer k, and estimates the conditional probability that a test observation x_0 belongs to a class j, based on what fraction of the k nearest neighbors belong to j. The mathematical representation is:

$$Pr\left(Y = j \mid X = x_0\right) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I\left(y_i = j\right),$$

where N_0 is the set of k-nearest neighbors, and $I(yi = j)$ is a dummy variable which evaluates to 1 when a given observation in N_0 is of class j, and to 0 otherwise.

Unlike the case for random forests, hyperparameter tuning can greatly impact the performance of the kNN algorithm. Most important is the choice of k. While setting k too low results in a classifier of low bias and high variance, setting k too high will conversely lead to a model of low variance but high bias. It is therefore important to find an optimal k which balances the two. [4]

There are several ways of finding optimal hyperparameters. The most commonly used method is grid search, which essentially works by training a classifier for several different combinations of the parameters of interest, and finding which hyperparameter setting leads to the highest accuracy. To increase the accuracy of our prediction, we can furthermore combine grid search with k-fold cross validation. K-fold cross validation works by having the dataset split into k subsets called folds (where k is usually 5 or 10), and subsequently training a sequence of models on these folds. This is done by training the first model on the first fold (with remaining folds used as the training set), the second model on the second fold, and so on. [5]

Running a grid search with 5-fold cross-validation, over a parameter grid where the range of k is 1 to 25, we find that the model reaches the highest accuracy when $k = 3$, so this is the k we will use for training the kNN.

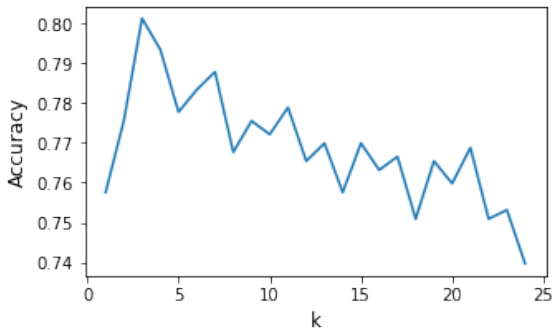

Figure 4: kNN accuracy w.r.t. k

3.3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is one of the simplest methods for stochastic optimization. Given a hypothesis class $\mathcal{W}$ and a training set T, the goal is to find a predictor $\mathbf{w}$ with an expected loss $F(\mathbf{w})$ that is close to optimal over $\mathcal{W}$. The algorithm is iterative, and works by initializing $\mathbf{w_1}$ to 0, and for each iteration $\{i = 1, 2, ...\}$ get a random estimate $\hat{\mathbf{g}}_\mathbf{i}$ of a subgradient of $F(\mathbf{w_i})$, such that $\mathbb{E}\hat{g}_i = g_i$. Each iteration $\mathbf{w_i}$ is then updated as follows:

$$\mathbf{w}_{i+1} = \prod_{\mathcal{W}} (\mathbf{w}_i - \eta_i \hat{\mathbf{g}}_i),$$

where η_i is step-size parameter, and $\prod_{\mathcal{W}}$ is the projection on $\mathcal{W}$. The output is a sequence of points $\mathbf{w_1}, \mathbf{w_2}, ...$, from which a final estimate with a function value close to F's minima can be obtained. [6]

3.4 Logistic Regression

Logistic regression is a mathematical modeling approach that can be used for describing the relationship between one or more variables X_k, and a dependent variable z. Mathematically, the logistic model can be written as

$$z = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k,$$

where α and β_1 represent unknown parameters. The logistic function can then be defined as

$$f(z) = \frac{1}{1+\mathrm{e}^{-z}} = \frac{1}{1+\mathrm{e}^{-(\alpha+\sum \beta_i X_i)}}$$

Considering our data, we have observed independent variables $X_1 = AGE$, $X_2 = GENDER$, and $X_3 = CLASS$ (where $CLASS$ denotes the ticket class) on a number of passengers, for whom we furthermore have a determined survival status 1 for "survived" and 0 for "not survived". Our logistic model can therefore be written as

$$P(Survived = 1|X_1, X_2, X_3) = \frac{1}{1+\mathrm{e}^{-(\alpha+\beta_1 \mathrm{AGE}+\beta_2 \mathrm{GENDER}+\beta_3 \mathrm{CLASS})}},$$

where we want to estimate the unknown parameters α, β_1, β_2, and β_3, and subsequently use the estimated parameters to predict the survival status of passengers in the test set. [7]

3.5 Support Vector Machines

Support Vector Machines (SVMs) are classifiers which map input vectors into a high-dimensional feature space through some pre-chosen mapping. Our objective then is to find the optimum linear hyperplane which best separate the two classes (in our case, survivors and non-survivors).

The classification function can be defined as

$$f(\mathbf{x}) = sign\,(\mathbf{w}'\phi(\mathbf{x}) + b),$$

where $\mathbf{x}$ is the input vector, ϕ is the feature function by which $\mathbf{x}$ is mapped into the high-dimensional feature space, $\mathbf{w}$ is the vector of weights, and b is the bias.

An approach to finding the optimum plane is to maximize the distance between the classes' respective supporting hyperplanes, by pushing the planes apart until they bump into the small number of data points in the training set that lie exactly on the borders of the margins. These data points are called the support vectors. The mapping by the feature function ϕ can be computed

through kernel functions, and the solution to the classification problem is then a weighted sum of kernels evaluated at the support vectors. [8] We will use a linear kernel in our implementation.

4 Evaluation

There are many evaluation metrics for machine learning methods. Apart from accuracy, which is the amount of correct predictions out of all the predictions, we will furthermore consider the confusion matrix. The confusion matrix is a common evaluation metric when working with machine learning classifiers, and it is essentially a representation of the true positives TP, false negatives FN, false positives FP, and true negatives TN. With these values, we can calculate the precision, recall, and subsequently the F-score: [9]

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$F - score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

In words, the precision can be described as the frequency with which positive identifications were actually correct, while recall is the frequency with which actual positives were correctly identified. The F-score is intended to combine the precision and recall to obtain a single measure of search effectiveness. [10]

All algorithms were tested on the given dataset which was already split into approximately 70% training data and 30% testing data. The performance of all models was furthermore estimated with a 10-fold cross validation.

4.1 Random Forest

The average accuracy of the random forest model was approximately 81.59%, with a standard deviation of 2.45%. In Figure 5 we see the confusion matrix averaged over the 10 folds.

On average, 482 passengers were correctly classified as having died (true negatives), 67 passengers were wrongly classified as having survived (false positives), 96 passengers were wrongly classified as having died (false negatives), and 246 passengers were correctly classified as having survived (true positives). The precision was therefore approximately 78.5%, while the recall was about 72% - so the F-score was slightly over 75%.

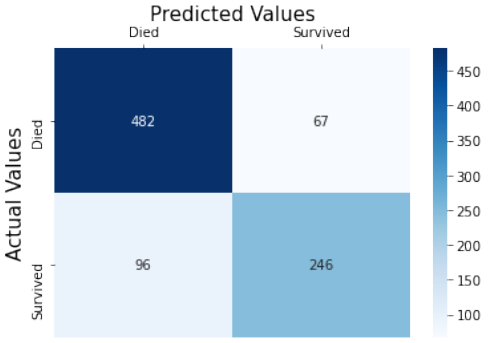

Figure 5: Random Forest Confusion Matrix

4.2 kNN

For the kNN model with k=3, the average accuracy was about 78.5%, and the standard deviation was 3.473%. Figure 6 shows the confusion matrix averaged over the 10 folds.

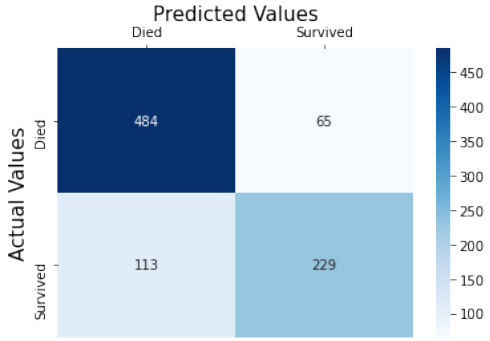

Figure 6: kNN Confusion Matrix (with $k = 3$)

We see that it is quite similar to the confusion matrix of the Random Forest, with 484 true negatives, 65 false positives, 113 false negatives, and 229 true positives. The precision was calculated to be around 78%, and the recall 67%. The F-score was therefore 72%.

4.3 Stochastic Gradient Descent

Our SGD algorithm performed with an average accuracy of 63.9%, and standard deviation 14.267%. Found in Figure 7 is the confusion matrix.

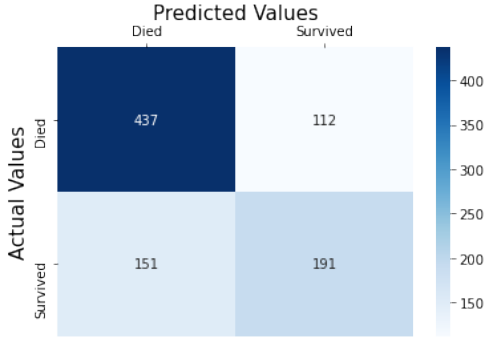

Figure 7: SGD Confusion Matrix

This confusion matrix deviates quite a bit from the previous ones, as the true negatives dropped down to 437, the false positives almost doubled to 112, the false negatives went up to 151, and the true positives went down to 191. This means a precision of 63%, a recall of less than 56%, and a resulting F-score of just above 59%.

4.4 Logistic Regression

Running the logistic regression algorithm resulted in an average accuracy of 77.66%, with a standard deviation of 4.268%. See Figure 8 for the confusion matrix.

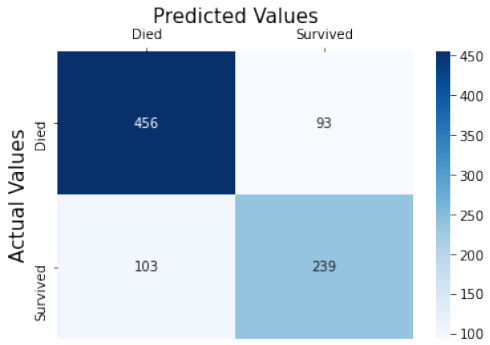

Figure 8: Logistic Regression Confusion Matrix

We can see that there was an average of 456 true negatives, 93 false positives, 103 false negatives, and 239 true positives. The precision was 72%, the recall was 70%, and the F-score was therefore 71%.

4.5 Support Vector Machines

Our final algorithm, the linear SVM, gave an average accuracy of 76.3%, with standard deviation 6.3%. Figure 9 shows the confusion matrix.

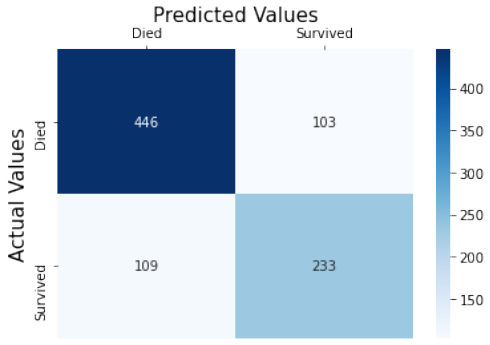

Figure 9: SVM Confusion Matrix

The average number of true negatives turned out to be 446, the false positives 103, the false negatives 109, and the true positives 233. The average precision was then 69.3%, while the recall was 68.1%. The resulting F-score was thereby approximately 68.7%.

5 Conclusion

The algorithm that performed the best in terms of average accuracy was the Random Forest model, as seen in the error bars of Figure 10.

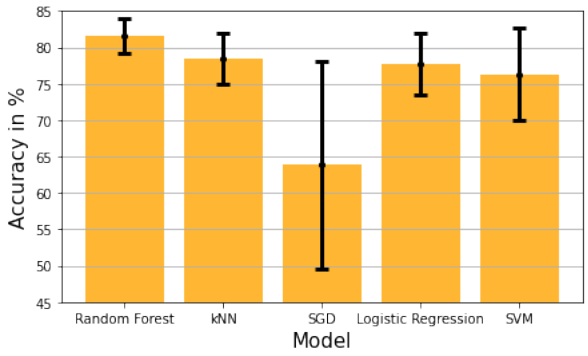

Figure 10: Standard deviation error bars w.r.t. accuracy

In Figure 11, we see how the algorithms performed in the areas of precision, recall, and F-score. The Random Forest algorithm outperformed the other algorithms in all three categories.

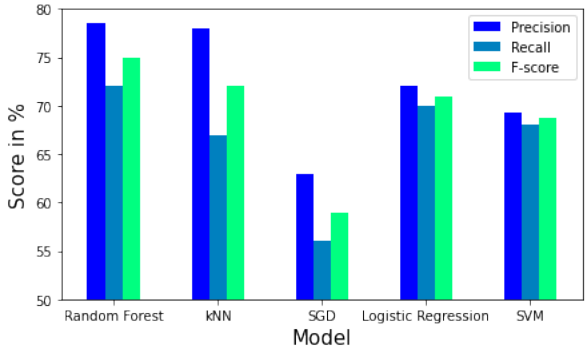

Figure 11: Algorithm comparison w.r.t. precision, recall, F-score

As such, we can conclude that the model which performed the best in this particular implementation was the Random Forest algorithm. However, we cannot make any claims on whether this is the case in general for this classification task, as the standard deviations - especially that of the SGD - indicate that repeating the experiment could result in another algorithm taking the lead. For future research, it would be interesting to investigate why the performance of the SGD indeed fluctuates so greatly. In terms of improvements to this research, it would have been optimal to compute the feature importance, rather than intuitively decide which features to train the classifiers on. Moreover, one should further explore hyperparameter tuning options for all chosen algorithms, rather than only tune the kNN as was done in this report.

References

[1] Ying X. 2019. "An Overview of Overfitting and its Solutions". Journal of Physics: Conference Series 1168(2), p. 4. https://doi.org/10.1088/1742-6596/1168/2/022022

[2] Brieman L. 2001. "Random Forests". Machine Learning 45(1), pp. 5–6. https://doi.org/10.1023/A:1010933404324

[3] Probst P., Wright M.N., Boulesteix A-L. 2019. "Hyperparameters and tuning strategies for random forest". WIREs Data Mining and Knowledge Discovery 9(3):e1301, p. 14. https://doi.org/10.1002/widm.1301

[4] James G., Witten D., Hastie T., Tibshirani R.: "Statistical Learning". In: An Introduction to Statistical Learning. Springer Texts in Statistics, vol 103, pp. 39–40. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-7138-7_2

[5] Müller A.C., Guido S. "Model Evaluation and Improvement". In: Introduction to Machine Learning with Python: A Guide for Data Scientists. Sebastopol, CA: O'Reilly Media, Inc. (2016), pp. 254 – 265.

[6] Rakhlin A., Shamir O., Sridharan K. 2011. "Making gradient descent optimal for strongly convex stochastic optimization". Computing Research Repository, pp. 1 – 3. https://arxiv.org/abs/1109.5647

[7] Kleinbaum D.G., Klein M.: "Introduction to Logistic Regression". In: Logistic Regression. Statistics for Biology and Health, pp. 4–10. Springer, New York (2010). https://doi.org/10.1007/978-1-4419-1742-3_1

[8] Maroco J., Silva D., Rodrigues A. et al. 2011. "Data mining methods in the prediction of Dementia: A real-data comparison of the accuracy, sensitivity and specificity of linear discriminant analysis, logistic regression, neural networks, support vector machines, classification trees and random forests". BMC Research Notes 4(299), pp. 3–4. https://doi.org/10.1186/1756-0500-4-299

[9] Olson D.L., Delen D. "Performance Evaluation for Predictive Modeling". In: Advanced Data Mining Techniques, pp. 137–138. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-76917-0_9

[10] Powers D.M.W. 2015. "What the F-measure doesn't measure: Features, Flaws, Fallacies and Fixes". arXiv preprint, p. 1. https://arxiv.org/abs/1503.06410