ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Final project report

Jerzy Pawlik 12242751

July 2023
Klagenfurt University

1 Project description

The project I completed for this course involved participating in a Kaggle Titanic competition.[1] The competition's objective was to use machine learning to create a model that could make accurate predictions for 418 passengers regarding who would survive the ship catastrophe. Each competitor received a training dataset containing information about 891 passengers and their fate in the catastrophe, along with a separate dataset of 418 passengers for making predictions. Both datasets were based on real Titanic passenger data. After submitting the results, competitors were ranked on a leaderboard.
In my solution, I primarily focused on preparing the data to identify the most critical factors influencing survival while excluding factors that only introduced noise to the data. Subsequently, I trained three different machine learning models using this data, evaluated their performance, and finally used the best model to make predictions and submit my results.

2 Motivation

This was my first project in machine learning, so my motivation was to just choose something which is good for getting started. As we know from history, in Titanic disaster passengers did not have equal chances of survival. They were segregated by social class, with first-class passengers having access to more lifeboats and being given priority during the evacuation. Additionally, crew members were instructed to prioritize the evacuation of women and children, which also had an impact on survival rates. That is why the ship disaster was a perfect case for me, to experiment with different supervised machine learning methods. The survival rates were mostly based on a few main factors and having the information about that factors among passengers, and whether they survived or not is everything what we need to create the model. Additionally opportunity to learn history by analysing historical data was also a plus.

3 Data

As mentioned earlier, each competitor received a training dataset containing information about 891 passengers (see Figure 1). The training set consists of 12 columns, encompassing numerical, categorical, and qualitative data. However, some columns also contain missing values.

Here is the list of all the columns:

- **PassengerId**
- **Survived** - "1" means that the passenger has survived
- **Pclass** - class of the ticket
- **Name -** name and title of the passenger
- **Sex -** passenger sex
- **Age -** age in years
- **SibSp -** number of siblings or spouses aboard the Titanic

1. Will Cukierski, *Titanic - Machine Learning from Disaster*, 2012, https://kaggle.com/competitions/titanic.

- **Parch -** number of parents or children aboard the Titanic
- **Ticket -** the ticket number
- **Fare -** the passenger fare
- **Cabin -** cabin number (encoded as letters and numbers)
- **Embarked** - port of embarkation. There were 3 ports of embarkation, each one is denoted by one letter: C = Cherbourg, Q = Queenstown, S = Southampton

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figure 1: overview on training data

4 Theoretical part

The Titanic competition is quite popular, and there is a wealth of studies and tutorials available online. Many different machine learning algorithms have been employed to solve it, with some outperforming others. For instance, I came across a paper that compared the outcomes of 10 different machine learning algorithms,[2] where the Kernel Support Vector Machine model achieved the best score. Similarly, another notebook compared 9 algorithms,[3] and in that case, Random Forest emerged as the winner. Interestingly, Random Forest also performed well in the first solution with the Kernel Support Vector Machine model. Because I was already familiar with the concept of decision trees, I decided to use the that algorithm for my project. Additionally, I utilized two other supervised machine learning algorithms covered during the lectures, namely logistic regression and K nearest neighbors. For some of the models I performed hyperparameter tuning what I have also learned from lectures, but I will cover this in the implementation part

The most challenging and creative aspect of this project was not merely creating the models but identifying and selecting the most relevant features to use in the model. As I do not know any definitive rule for that, I analyzed other solutions [4],[5] and combined the findings with my own intuition to distinguish new features in the data. The subsequent step involved selecting the best features among them. To achieve this, I analyzed correlations and employed linear regression. The linear regression was discussed during the lectures as a supervised machine learning method to make a predictions, but it also allows to find the most important independent variables, that influence the depended variable (in this case "survived").

2. Charles-Antoine de Thibault, "10 Machine Learning Models to predict Titanic Survival," January 2017, https://rstudio-pubs-static.s3.amazonaws.com/348904_1a80d04f436f4a41892179fbae1270c9.html.

3. Manav Sehgal, "Titanic Data Science Solutions," Kaggle, February 2019, https://www.kaggle.com/code/startupsci/titanic-data-science-solutions/notebook?scriptVersionId=10431564.

4. Sumit Mukhija, "A beginner's guide to Kaggle's Titanic problem," Towards Data Science, June 2019, https://towardsdatascience.com/a-beginners-guide-to-kaggle-s-titanic-problem-3193cb56f6ca.

5. Sehgal, "Titanic Data Science Solutions."

5 Implementation

5.1 preparing the data

The first thing that I did after reading the data was extracting titles from a “name” column and displaying them on histogram (see figure 2).
Furthermore, I performed a count of all the not assigned values per column (see Figure 3).

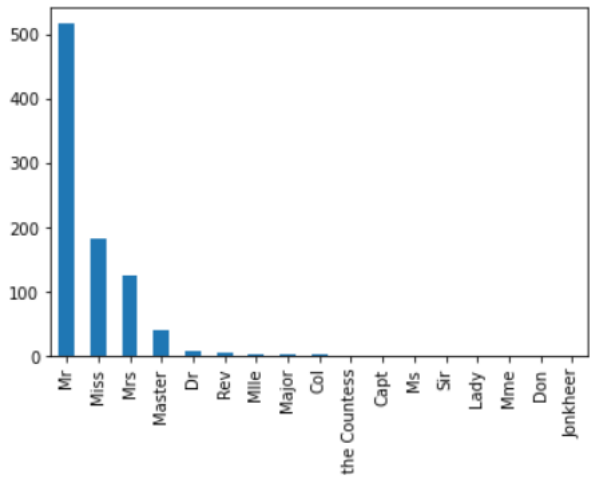

Figure 2: titles in a training dataset

```
PassengerId:            0/891   0/418
Pclass:         0/891   0/418
Name:           0/891   0/418
Sex:            0/891   0/418
Age:            177/891 86/418
SibSp:          0/891   0/418
Parch:          0/891   0/418
Ticket:         0/891   0/418
Fare:           0/891   1/418
Cabin:          687/891 327/418
Embarked:               2/891   0/418
title:          0/891   0/418
```

Figure 3: counting not assigned values in training and test set

5.2 cleaning the data

It became evident that there were numerous rare titles that only added noise to the data. To address this issue, I decided to replace these infrequent titles with the titles that were most similar to them (see Figure 4). For instance, title “Sir” was an aristocratic title for men, but I treated them as ”Mr” (adult man)

```python
def changeTitle(title):
    if str(title) == "Mlle" or str(title) == "Mme":
        title = 'Miss'
    elif str(title) == 'Dona' or str(title) == 'Lady' or str(title) == 'Ms' or str(title) == 'the Countess':
        title = 'Mrs'
    elif str(title) == 'Major' or str(title) == 'Col' or str(title) == 'Capt':
        title = 'officer'
    elif str(title) == 'Jonkheer':
        title = 'Master'
    elif title == 'Sir' or title == 'Don':
        title = 'Mr'
    return title
```

Figure 4: function for changing the title

Furthermore I dropped the columns that in my opinion did not contribute any meaningful information or had too many missing values, filled the NA values in other columns and scaled the data (see figure 5). Because I wanted to use logistic regression that requires numerical data and KNN, that has to compute distances I employed one-hot encoding for titles column. This have increased data dimensionality, but for this two algorithms, categorical data has to be encoded.

```
def clean(data):

    #dropping uneeded columns:
    data = data.drop(["PassengerId","Name","Ticket","Cabin"], axis=1)

    #Encoding values of Sex column
    encoder = LabelEncoder()
    data['Sex'] = encoder.fit_transform(data['Sex'])

    #filling NA in column fare with median fare:
    data["Fare"].fillna(data["Fare"].median(), inplace=True)

    #filling NA in column age, so that it would be similarly distributed to already existing data:
    mean = data['Age'].mean()
    std = data['Age'].std()
    null_count = data['Age'].isnull().sum()
    samples = np.random.normal(loc=mean, scale=std, size=null_count)
    data.loc[data['Age'].isnull(), 'Age'] = samples

    #changing titles, to decrese dimensionality
    for i in range(len(data)):
        title = data.title[i]
        title = changeTitle(title)
        data.title[i] = title

    data = pd.concat([data, pd.get_dummies(data["title"])], axis=1)
    del data["title"]

    #I map mananualy the ports based on their order
    mappings = {'S':int('0'),'C':int('1'),'Q':int('2')}
    data.Embarked.fillna("C")
    data["Embarked"] = custom_label_encoder(data["Embarked"], mappings)

    return data

data = clean(data)
test = clean(test)
```

Figure 5: function for cleaning the data

5.3 data analysis

The previous steps have lead the data to have 15 columns, and the next step was to limit the number of dimensions by selecting the best features. To improve the model's outcomes, I aimed to extract more meaningful features from the existing ones and make substitutions. "SibSp" and "Parch" columns seemed an interesting choice for that.My hypothesis was that people traveling alone might have lower chances of survival since they had no one to take care of them. To validate this, I separately plotted survival rates based on these columns. In both cases for "0" the survival rates appeared to be significantly lower.

However, I made another interesting observation: for bigger families, the survival rates were also low. I explained this as a potential challenge in evacuating effectively for larger groups due to their need for more attention to prevent splitting up. This seemed reasonable for me, but of course there may be other reasonable ways to explain it. Consequently, I created a new column named: "relatives" which was a sum of "SibSp" and "Parch" and plotted survival rates for each number of relatives (see figure 6). All patterns observed before also appeared to be true there.

For further analysis, I crated a frequency histogram for the new column (see figure 7) This revealed that big families were relatively rare but not so infrequent as to be limited to only single cases. In the work that I referenced to earlier[6] this columns were also combined and studied, which led to distinguishing "Is_Alone" category. However, my analysis was slightly different, and based on the information I obtained, I decided to include an additional "bigFamily" category for the number of relatives greater than 3.

6. Sehgal, "Titanic Data Science Solutions."

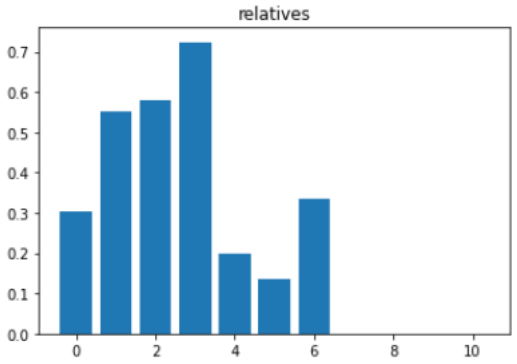

Figure 6: relatives number and survival rates

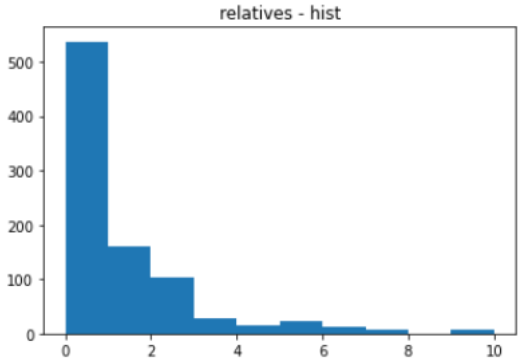

Figure 7: relatives column frequency histogram

Apart from that, I conducted some analysis using plots for the "fare" and "class" columns to understand how they were related to each other and gain better insights into the data. Since these columns were correlated, I hoped to identify one of them as a candidate for removal later in the feature selection process.

5.4 features selection

To perform feature selection, I began by displaying a correlation table. Then, by creating linear regression models that excluded some features, I tried to find the best feature subset. I selected features to be excluded based on their P-value in linear regression, correlation with other features (if one feature may be used to explain the other, there may be no sense to keep both of them in the model), my reasoning and intuition (If I have a columns with titles, there may be no reason to have a sex column because this information is already repeated in titles). Ultimately, I chose the feature subset that had slightly lower R-squared and Adj. R-squared values than the full feature subset but was significantly smaller (see Figure 8).

	Survived	Pclass	Age	Miss	Mr	Mrs	Rev	alone	bigFamily
0	0.0	1.0	0.416020	0.0	1.0	0.0	0.0	0.0	0.0
1	1.0	0.0	0.577118	0.0	0.0	1.0	0.0	0.0	0.0
2	1.0	1.0	0.456294	1.0	0.0	0.0	0.0	1.0	0.0
3	1.0	0.0	0.546912	0.0	0.0	1.0	0.0	0.0	0.0

Figure 8: dataset used to feed a model

5.5 creating and evaluating the models

For training the models I divided the dataset into training and validation set in proportion 4-to-1. For hyperparameter tuning I used GridSearchCV class form Python scikit-learn library. This class enables performing grid search with cross validation through all the given values of parameters.

5.5.1 Random forest

Random forest is an algorithm that creates multiple decision trees that use random subsets of features from a dataset to make decision. the final decision is made by voting of the whole forest. The hyperparateres there that I decided to tune here are as follows:

- max_depth - controls the maximum depth of each individual decision tree in the forest. A deeper tree can learn complex relationships in the data, but it also increases the risk of overfitting. A shallow tree may not capture all the patterns in the data.
- criterion - criterion that decides how to make a split while building the tree
- class_weight - is used to address class imbalance in the dataset. By setting the class_weight parameter, different weights can be assigned to different classes to give higher importance to the minority class during training. This helps the model to focus more on correctly predicting the rare class and can improve overall performance, especially in imbalanced datasets.

After fitting the model, I obtained a satisfying accuracy (see Figure 9). Additionally, the values of precision and recall were also good, indicating that the model performed well in both correctly identifying positive cases and minimizing false positives.

```
Accuracy = 0.7932960893854749
Recall (rate of survived people predicted as survived) = 0.7681159420289855
Precission (how many people predicted as survived have actually survived) = 0.7162162162162162
[[89 16]
 [21 53]]
```

Figure 9: random forest scores

5.5.2 Logistic regression

I wanted to use this model as a comparison to random forest, so I did not tune any parameters and expected that the results will be worse, but the results of the evaluation were surprisingly good and very similar to random forest (see figure 10)

```
Accuracy = 0.7988826815642458
Recall (rate of survived people predicted as survived) = 0.7794117647058824
Precission (how many people predicted as survived have actually survived) = 0.7162162162162162
[[90 15]
 [21 53]]
```

Figure 10: logistic regression scores

5.5.3 K nearest neighbours

This model was discussed during the classes so I will not explain how it works and only list the parameters that I tuned

- n_neighbors - to small values my lead to underfitting and to high to overfitting
- metric - distance measures between data points
- weights - when set to distance, the votes will be weighted according to distance from the data point.

The accuracy here also appeared be similar to the other models, but the difference between precision and recall was bigger, so the model was biased towards predicting as survived (see figure 11)

```
Accuracy = 0.7988826815642458
Recall (rate of survived people predicted as survived) = 0.8166666666666667
Precission (how many people predicted as survived have actually survived) = 0.6621621621621622
[[94 11]
 [25 49]]
```

Figure 11: KNN scores

6 Conclusions

The accuracy scores obtained on the validation sets in the two notebooks I mentioned earlier were better than mine (around 85% while mine were near 80%). However, I do not know their scores on the test set for the final submission. When I looked at the competition leaderboard, scores of around 85% were in the top 1% of submissions. The author of "Titanic Data Science Solutions"[7] reported scoring 3,883 out of 6,082 competition entries that were there back then. The predictions from the "Beginner's Guide"[8] notebook were in the top 8% of submissions. Therefore, their models might be a bit overfitted, and their real scores might be smaller.

For the paper from Charles-Antoine de Thibault,[9] the best entry had a score of 0.78468 and differed only slightly from the score obtained during training. My best entry, which was the random forest, also had a similar accuracy of 0.78708. This score was satisfying for me and also did not differ significantly from the score obtained during training. The logistic regression and KNN models were also not much worse, with scores of 0.78229 and 0.78468, respectively.

Although task in Titanic competition seems to be pretty simple at first, there are numerous aspects that can be explored there. I am aware that I have not used all the information in the data and there are a lot of things that can be done in future works. For example I did not investigate "cabin" and "ticket" columns at all. I also have not investigated all the correlations and certainly there can be a lot of hypotheses explaining this correlations that can be tested in the future.

7. Sehgal, "Titanic Data Science Solutions."
8. Mukhija, "A beginner's guide to Kaggle's Titanic problem."
9. Thibault, "10 Machine Learning Models to predict Titanic Survival."

References

Cukierski, Will. *Titanic - Machine Learning from Disaster*, 2012. https://kaggle.com/competitions/titanic.

Mukhija, Sumit. "A beginner's guide to Kaggle's Titanic problem." Towards Data Science, June 2019. https://towardsdatascience.com/a-beginners-guide-to-kaggle-s-titanic-problem-3193cb56f6ca.

Sehgal, Manav. "Titanic Data Science Solutions." Kaggle, February 2019. https://www.kaggle.com/code/startupsci/titanic-data-science-solutions/notebook?scriptVersionId=10431564.

Thibault, Charles-Antoine de. "10 Machine Learning Models to predict Titanic Survival," January 2017. https://rstudio-pubs-static.s3.amazonaws.com/348904_1a80d04f436f4a41892179fbae1270c9.html.