Final Report Titanic Disaster

Christoph Hagenauer*, Shishir Nath*

May 30, 2025

Contents

*University of South Carolina Beaufort

1 Introduction

1.1 The titanic disaster

The Titanic was a massive, luxurious British passenger ship known as the largest and "unsinkable" ship of its time. On its very first voyage in April 1912, it hit an iceberg late at night, which tore open the hull and caused it to flood and sink within a few hours. The ship did not have enough lifeboats for everyone, and because of poor emergency planning, over 1,500 people died. The disaster shocked the world and led to big changes in ship safety rules.

1.2 What is this project?

In our project we engineered an AI to predict whether or not a passenger lived in the titanic disaster. For this purpose, we used a dataset hosted on Kaggle in the *Titanic - Machine Learning from Disaster* challenge. Kaggle is a platform for datasets and AI challenges. The dataset is part of the Titanic challenge in which we, as developers, engineer a machine learning model to predict whether or not a passenger lived in the titanic disaster.

1.3 Collaboration approach

Our team has two members: Christoph Hagenauer and Shishir Nath. We collaborated closely to exchange ideas, and implement them. We utilized Microsoft Teams to communicate outside of personal meetings. We established our baseline, analyzing the dataset and preparing the data for machine learning, together in person. Once we had our baseline, we both experimented with various models and performed hyperparameter tuning to optimize our models.

1.4 Timeline and Milestones

We divided the project into four sprints, each sprint lasting one week:

- Sprint 1 (March 24 – March 30): Get a deeper understanding of the data and impute any missing values; prepare the data for machine learning and neural networks
- Sprint 2 (March 31 – April 6): Try various machine learning models available in scikit-learn. Evaluate them and make our first submission to the Titanic challenge.
- Sprint 3 (April 7 – April 13): Build our neural network using Keras; make another submission and evaluate the results; perform hyperparameter tuning as necessary
- Sprint 4 (April 14 – April 20): Giving an in-progress presentation and performing hyperparameter tuning.
- Sprint 5 (April 21 – April 27): Finish the project by writing a report and give our final evaluation of the data munging techniques used and evaluate the performance of our models.

1.5 Technical Approach

We used python as our programming language. Specifically, we used pandas to handle our data, keras to build neural networks, seaborn to visualize data and metrics of our models such as the roc curve and confusion matrix, numpy to perform matrix multiplications and to feed the data to the models, and scikit-learn to perform data preprocessing (Imputation, splitting the data for training and testing) and for machine learning models.

2 Data Wrangling

Each sample in our dataset represents one person. Each person has data about their ticket class, name, gender, age, the number of family members on board, ticket number, ticket fair, cabin number, and the port they embarked the ship. However, some passengers have missing data, therefore we will use

imputation to fill these missing entries. The dataset we used to train our model has a total of 891 entries. For the submission to the Titanic challenge, we have to predict the outcome for 418 passengers. In Appendix A we share a sample of the available data.

2.1 Statistic in the data set

For our numerical analysis we decided to use the five number statistic. In Table 1 we share the results. Looking at this table, we can observe the following:

- At least 50% of the people died in the disaster.
- More than 50% of the people were traveling in 3^{rd} class.
- Most people were around the age of 28, with the 25 percentile being at 20.12 and the 75 percentile being at 38 years old.
- Looking at the SibSp and Parch columns, we can observe that most people were traveling by themselves.
- The fare reflects the class people were traveling in.

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.00	891.00	714.00	891.00	891.00	891.00
mean	0.38	2.31	29.70	0.52	0.38	32.20
std	0.49	0.84	14.53	1.10	0.81	49.69
min	0.00	1.00	0.42	0.00	0.00	0.00
25%	0.00	2.00	20.12	0.00	0.00	7.91
50%	0.00	3.00	28.00	0.00	0.00	14.45
75%	1.00	3.00	38.00	1.00	0.00	31.00
max	1.00	3.00	80.00	8.00	6.00	512.33

Table 1: 5 Number Statistic

For our categorical data we observed the following:

- 55% of people traveled in 3^{rd} class, while about 25% traveled in 1^{st} and 2^{nd} class each.
- The majority of people embarked the ship in Southampton accounting for 72.5%, while 18.9% embarked in Cherbourg, and only 8.6% embarked in Queenstown.
- In our training data only 38.38% of people survived the disaster, which is a little higher than the overall survival, which was at 32%.

Finally, we did not use the cabin information due to too many values. Furthermore, we did not perform any feature engineering on the name due to time constraints, nor did we perform any feature engineering on the ticket.

We imputed missing categorical data with the most-frequent strategy and numerical data with the mean. After imputation, the dummies were created for categorical data with more than two classes, and all numerical features were scaled using the Standard Scaler from scikit-learn.

2.2 PCA

For our PCA analysis we decided to use six components as they explain over 90% of variance in the data. We see that Sex, Age, and Fare have the highest variance within the data. This aligns with the observation during the data analysis, where we saw that Age, Fare, and Sex have a significant influence on survivability. These findings are presented in Figure 1. After experimenting with using different dimensions for our models, we found that performing PCA significantly reduced our models' performance.

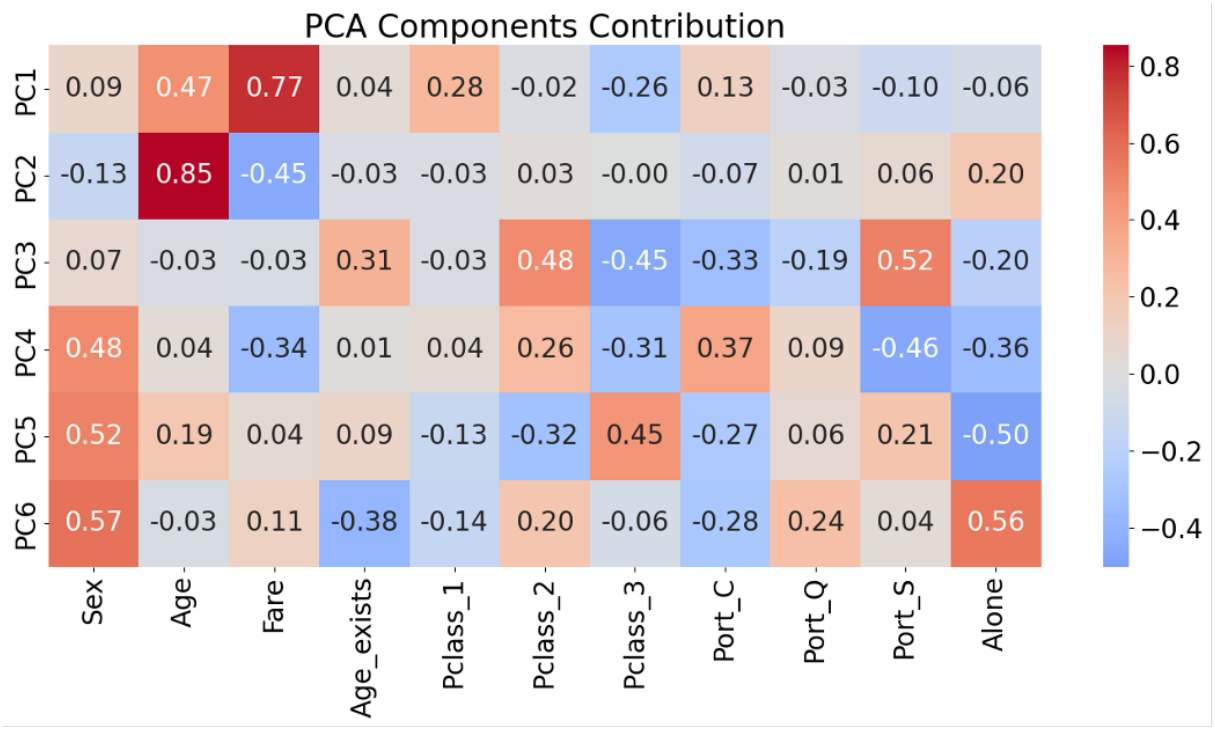

Figure 1: PCA visualization

3 Machine Learning

Resulting out of the imbalance in the target variable, survival, we calculated the weights to use in the machine learning algorithms to detect that imbalance. However, we saw decreasing results when using the weights in our training process, which is most likely due to the fact that we want our model to predict more deaths since more people died than lived in the disaster.

3.1 Simple Perceptron

For our simple perceptron we performed a one layer calculation. The dot product between the weights and each input was calculated and afterward the relu activation function was applied. For details, please see Appendix B. The simple perceptron scored quite high, considering its simplicity, as shown in Figure ref.

3.2 Machine Learning with Scikit-Learn

For machine learning, we used scikit-learn. We performed hyperparameter tuning using Halving Grid Search CV, which after each iteration only considered the best half-performing parameters and therefore is ideal for a large search space. In total, we considered three machine learning algorithms, one linear model with Logistic Regression, and two ensemble models, Gradient Boosting Classifier, and Random Forest Classifier. Ensemble models have the advantage of detecting more advanced patterns in the data as they use a base estimator and then built another estimator on top of that, each estimator detecting different features.

3.3 Neural Network

For our neural network, we used keras, an API to easily build neural networks. We performed hyperparameter tuning using the hyperband search algorithm. This algorithm optimizes the search by eliminating the worst performing parameters after each round. For our search we specified four activation functions (relu, tanh, squarelus, and selu). Each layer had the choice from 4 to 4096 parameters in a step size of log(2). For code details, please see Appendix D.

3.4 Model Evaluation

Looking at our metric plots, we found that the models performed very evenly. The best performing model in our training is Logistic Regression with an AUC score of 80.1%. However, Random Forest Classifier and Gradient Boosting Classifier performed almost the same achieving AUC of about 79%. The neural network performed the worst with an AUC score of only 77.3%. Especially considering the time required to train and tune the neural network the performance is underwhelming. However, we also want to point out that we are only training our models with less than 1000 samples, and neural networks excel with more data, while struggling with fewer samples. The metrics for the models are visualized in Figure 2. For the confusion matrix the true labels are on the y -xis and the predicted labels on the x-axis.

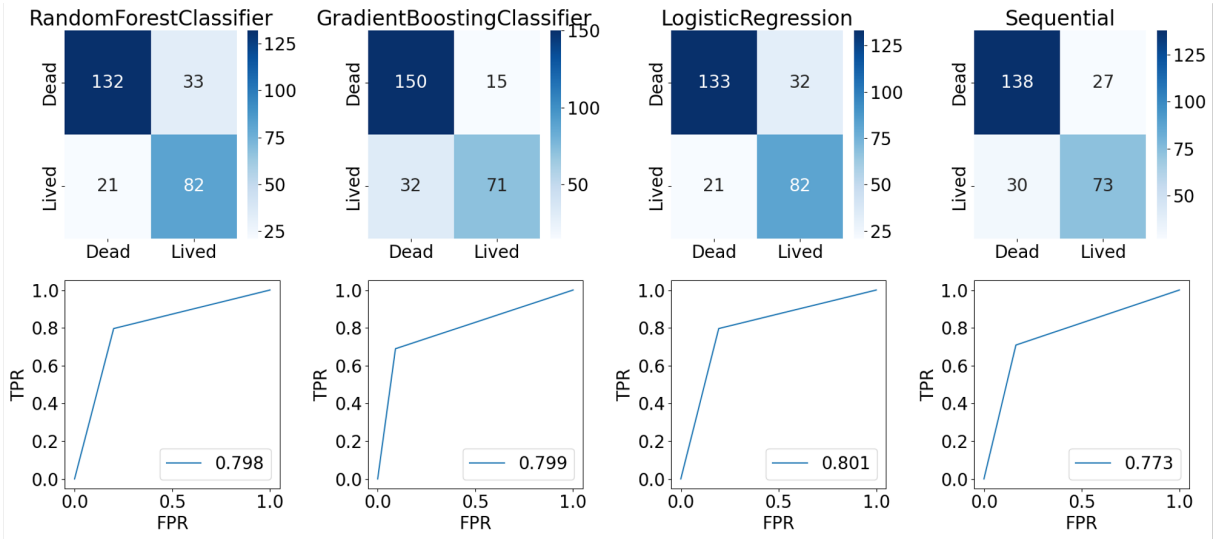

Figure 2: Model evaluatoin

4 Lessons learned

We conclude this project by presenting our takeaways from this competition. We saw that sometimes an easier approach such as the simple perceptron, and machine learning algorithms from scikit-learn are able to achieve the same if not a higher score than a neural network, which takes a lot longer to train and tune. Our biggest takeaway is that even the best model will only get slightly better results, because in the end everything depends on the data and how we as engineers feed it to the models.

A Sample Data

Survived	Pclass	Name	Sex	Age
0	3	Braund, Mr. Owen Harris	male	22
1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38
1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35
0	3	Allen, Mr. William Henry	male	35
0	3	Moran, Mr. James	male	-
0	1	McCarthy, Mr. Timothy J	male	54
0	3	Palsson, Master. Gosta Leonard	male	2
1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27
1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14
1	3	Sandstrom, Miss. Marguerite Rut	female	4
1	1	Bonnell, Miss. Elizabeth	female	58
0	3	Saundercock, Mr. William Henry	male	20
0	3	Andersson, Mr. Anders Johan	male	39
0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14
1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55
0	3	Rice, Master. Eugene	male	2
1	2	Williams, Mr. Charles Eugene	male	-

SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	A/5 21171	7.25	-	S
1	0	PC 17599	71.28	C85	C
1	0	113803	53.10	C123	S
0	0	373450	8.05	-	S
0	0	330877	8.46	-	Q
0	0	17463	51.86	E46	S
3	1	349909	21.07	-	S
0	2	347742	11.13	-	S
1	0	237736	30.07	-	C
1	1	PP 9549	16.70	G6	S
0	0	113783	26.55	C103	S
0	0	A/5. 2151	8.05	-	S
1	5	347082	31.27	-	S
0	0	350406	7.85	-	S
0	0	248706	16	-	S
4	1	382652	29.12	-	Q
0	0	244373	13	-	S

Table 4: A sample of the available data

B Simple Perceptron algorithm

```
import random
random.seed(42)
weights = [random.uniform(0,2), random.uniform(0,2), random.uniform(0,2),
            random.uniform(0,2), random.uniform(0,2), random.uniform(0,2),
            random.uniform(0,2), random.uniform(0,2), random.uniform(0,2),
            random.uniform(0,2), random.uniform(0,2)]
bias = random.uniform(0,2)
epochs = 1000
learning_rate = 0.1
def step_function(y_pred):
    if y_pred >= 0:
        return 1
    else:
        return 0
# Training the model
weighted = False
for epoch in range(epochs):
    x = X_train[random.randint(0, X_train.shape[0]-1)]
    y_true = y_train[random.randint(0, X_train.shape[0]-1)]
    y_pred = step_function(np.dot(x, weights) + bias)
    if (epoch) % 100 == 0:
        print(f'Epoch {epoch+1}: Prediction {y_pred}; Label {y_true}')
    for i in range(len(weights)):
        if weighted:
            if y_pred == 1:
                weights[i] = weights[i] + learning_rate *
                        (y_true - y_pred) * x[i]
            else:
                weights[i] = weights[i] + learning_rate *
                        (y_true - y_pred) * x[i]
        else:
            weights[i] = weights[i] + learning_rate * (y_true - y_pred) * x[i]
    if y_pred == 1 and weighted:
        bias = bias + learning_rate * (y_true - y_pred)
    elif y_pred == 0 and weighted:
        bias = bias + learning_rate * (y_true - y_pred)
    else:
        bias = bias + learning_rate * (y_true - y_pred)
print(f'Final weights and bias:
    {" ".join(str(i) for i in weights)} bias = {bias:.2f}\n')
y_pred = []
for X in X_test:
    y_pred.append(step_function(np.dot(X, weights)+bias))
print(len(y_test), len(y_pred))
# print(y_pred)

fig, ax = plt.subplots(2, 1, figsize=(6, 11))
ax[0].set_title("Simple Perceptron")
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Dead", "Lived"],
            yticklabels=["Dead", "Lived"],
            ax=ax[0])
print("Smiple Perceptron", classification_report(y_test, y_pred))
fpr, tpr, threshold = roc_curve(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred)
# ax[1] = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=auc_score)
sns.lineplot(x=fpr, y=tpr, label=f"{auc_score:.3f}", ax=ax[1], legend='brief')
ax[1].set_xlabel("FPR")
ax[1].set_ylabel("TPR")
ax[1].legend(loc='lower right')
plt.show()

y_pred = []
for X in X_submission:
    y_pred.append(step_function(np.dot(X, weights)+bias))
df_submission = pd.DataFrame(zip(test_data.PassengerId.copy(), y_pred),
    columns=["PassengerId", "Survived"])
df_submission.to_csv(f"test_prediction_Simple_Perceptron.csv", index=False)
```

C Machine Learning

```
fig, ax = plt.subplots(2, 4, figsize=(20, 9))
gbc = GradientBoostingClassifier()
lg = LogisticRegression(max_iter=10000, n_jobs=-1)
rf = RandomForestClassifier(n_jobs=-1)
classifiers = [rf, gbc, lg, hypermodel]
parameters_rf = { 'n_estimators':[100, 1000, 10000], 'criterion':('gini',
'entropy', 'log_loss'), 'max_depth':[2,4,8,16],
                 'class_weight':(None, class_weight)}
parameters_lg = {'penalty':('l2', None), 'dual':(True, False),
'fit_intercept':(True, False), 'class_weight':(class_weight, None),
'solver':('lbfgs', 'newton-cg', 'sag', 'saga')}
parameters_gbc = {'loss':('log_loss', 'exponential'), 'learning_rate':[0.1, 0.001], 'n_estimators':[100,
     10000],
'criterion':('friedman_mse', 'squared_error'), 'max_features': ('sqrt', 'log2')}
parameters = [parameters_rf, parameters_gbc, parameters_lg]
perform_search =  True
for col, classifier in enumerate(classifiers):
    if classifier != hypermodel:
        if perform_search:
            search = HalvingGridSearchCV(classifier, parameters[col], n_jobs=-1, cv=10)
            search.fit(X_train, y_train)
            y_pred = search.predict(X_test)
            y_submisson_predict = search.predict(X_submission)
        else:
            classifier.fit(X_train, y_train)
            y_pred = classifier.predict(X_test)
    else:
        y_pred = classifier.predict(X_test)
        y_pred = (y_pred[:, 0] > 0.445).astype(int)
        y_submisson_predict = classifier.predict(X_submission)
        y_submisson_predict = (y_submisson_predict[:, 0] > 0.445).astype(int)
    # Save prediction for competition
    df_submission = pd.DataFrame(zip(test_data.PassengerId.copy(), y_submisson_predict), columns=["
     PassengerId", "Survived"])
    df_submission.to_csv(f"test_prediction_{str(classifier.__class__.__name__)}.csv", index=False)
    # Print metrics
    print(str(classifier.__class__.__name__), classification_report(y_test, y_pred))
    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=["Dead", "Lived"],
                yticklabels=["Dead", "Lived"],
                ax=ax[0, col])
    ax[0, col].set_title(str(classifier.__class__.__name__))
    # ROC Curve and AUC score
    fpr, tpr, threshold = roc_curve(y_test, y_pred)
    auc_score = roc_auc_score(y_test, y_pred)
    sns.lineplot(x=fpr, y=tpr, label=f"{auc_score:.3f}", ax=ax[1, col], legend='brief')
    ax[1, col].set_xlabel("FPR")
    ax[1, col].set_ylabel("TPR")
    ax[1, col].legend(loc='lower right')
plt.tight_layout()
plt.show()
```

D Neural Network

```
set_random_seed(812)
def build_model(hp):
        hp_activation_layer_1 = hp.Choice('activation 1', values=['relu', 'tanh', 'squareplus', 'selu'])
        hp_activation_layer_2 = hp.Choice('activation 2', values=['relu', 'tanh', 'squareplus', 'selu'])
        hp_activation_layer_3 = hp.Choice('activation 3', values=['relu', 'tanh', 'squareplus', 'selu'])
        hp_activation_layer_4 = hp.Choice('activation 3', values=['relu', 'tanh', 'squareplus', 'selu'])
        hp_layer_1 = hp.Int('units 1', min_value=4, max_value=4096, step=2, sampling='log')
        hp_layer_2 = hp.Int('units 2', min_value=4, max_value=4096, step=2, sampling='log')
        hp_layer_3 = hp.Int('units 3', min_value=4, max_value=4096, step=2, sampling='log')
        hp_layer_4 = hp.Int('units 3', min_value=4, max_value=4096, step=2, sampling='log')

        model = Sequential()
        model.add(Input(shape=(X_train.shape[1],), name='Input_Layer'))
        model.add(Dense(hp_layer_1, activation=hp_activation_layer_1, kernel_regularizer=l2(0.01), name='
    hidden_1'))
        model.add(Dense(hp_layer_2, activation=hp_activation_layer_2, kernel_regularizer=l2(0.01), name='
    hidden_2'))
        model.add(Dense(hp_layer_3, activation=hp_activation_layer_3, kernel_regularizer=l2(0.01), name='
    hidden 3'))
        model.add(Dense(hp_layer_4, activation=hp_activation_layer_4, kernel_regularizer=l2(0.01), name='
    hidden_4'))
        model.add(Dense(1, activation='sigmoid', name='hidden_3_output'))
        loss = losses.BinaryCrossentropy()
        opt = optimizers.Adam(learning_rate=1e-2)
        model.compile(loss=loss, optimizer=opt, metrics=['accuracy'])

        return model

callback = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True, start_from_epoch=10)
lr_decay = ReduceLROnPlateau(monitor='loss', patience=1, verbose=0, factor=0.5, min_lr=1e-7)
hp = keras_tuner.HyperParameters()
hp_batch_size = hp.Int('batch_size', min_value=2, max_value=128, step=2, sampling='log')
tuner = keras_tuner.Hyperband(build_model,
                    objective=keras_tuner.Objective('val_accuracy', direction='max'),
                    max_epochs=100,
                    factor=10,
                    directory='Hyperband',
                    project_name='titanic',
                    overwrite=True)

categorical_y_train = y_train
categorical_y_test = y_test

tuner.search(X_train, categorical_y_train, batch_size=64, epochs=100, validation_split=0.2,
             callbacks=[callback, lr_decay], class_weight=class_weight)
```