CISC 5800 Project Report

Machine Learning from Disaster

Jingyan Xu[1] and Haoran Xue[2]

[1]Department of Computer and Information Sciences, Fordham University
[2]Department of Computer and Information Sciences, Fordham University

May 11, 2023

Abstract

This project report investigates the application of Random Forest and Support Vector Machines with Sequential Minimal Optimization algorithms on the Titanic survival dataset. The dataset contains information about passengers aboard the Titanic and whether they survived or not. In this report, we explore the implementation and performance of these two algorithms on the Titanic survival dataset. We also discuss the impact of different parameters on the performance of each algorithm.

Keywords
Machine Learning, Random Forest, Support Vector Machine, Sequential Minimal Optimization

1 Introduction

The RMS Titanic's sinking is a well-known maritime disaster that occurred during its first journey on April 15, 1912. The ship collided with an iceberg, resulting in the death of 1502 out of the 2224 people aboard. This catastrophic event was a significant shock to the global community and resulted in the implementation of improved safety protocols for ships. Our objective is to make accurate predictions about the survival of passengers in the Titanic disaster, utilizing machine-learning techniques, based on the data available on OpenML [1]. Factors such as ticket cost, age, gender, and social class will be taken into account to make these predictions.

As how machine learning algorithms are always implemented, we split the dataset into training set and testing set. We used the training set to build our model to generate predictions for the testing set. We were to make predictions for each passenger in the test set, indicating whether or not they survived the sinking of the Titanic.

One requirement of the project is to use two machine learning techniques, one can be taken from publicly available software and one must be implemented from the scratch. We developed multiple strategies to analyze and compare various machine learning techniques. By studying the outcomes of each approach, we can gain valuable insights into the problem. The techniques utilized in this project involve Random Forest Classifier using scikit-learn, a free Python machine learning library, and Support Vector Machine with a Sequential Minimal Optimization (SMO) implementation. Through these techniques, we aim to predict passenger survival by experimenting with different combinations of features.

Essentially, the problem at hand was a classification task based on a set of features. One method for prediction was to use Random Forest, while another was to utilize SMO algorithm to implement SVM classification. We developed the project in a two-track process, unlike many other machine learning projects where one classification technique are implemented upon another in order to achieve better result, the two

machine learning techniques were independent of each other. They would both use the same preprocessed-dataset, but the algorithms would be different in order to compare the results.

2 Dataset

The dataset was accessed through OpenML module which allows us to use and share datasets and tasks, run machine learning algorithms on them. We found that there were several datasets available named "Titanic", and we picked the one that has the most downloads up to the date of this report.

The Titanic dataset contains information about the survival status of individual passengers on the Titanic. The datasets provides acutal and estimated ages for around 80% of the passengers, but does not include data for the crew. The variables in the extracted dataset include pclass, survived, name, age, sibsp, parch, embarked, home.dest, room, ticket, fare, body, boat, and sex. Passenger class serves as one of the proxies for socioeconomic class and is categorized into first, second, and third. Age is measured in years, with some infants having fractional values. Fare is measured in pre-1970 British pounds. These three variables would play major role in our models.

The dataset consists of 14 variables. The Survived variable is a binary flag that shows if a passenger survived the voyage or not. The Pclass variable indicates the class of each passenger. The Name variable contains the name of each passenger. The Sex variable identifies the gender of each passenger. The Age variable represents the age of each passenger. The Sibsp and Parch variables indicate the number of siblings/spouses and parents/children aboard, respectively. The Ticket variable contains the ticket number issued to each passenger. The Fare variable represents the amount of money spent on the ticket. The Cabin variable indicates the category of cabin occupied by each passenger. The Embarked variable shows the port of embarkation. The Boat variable contains the identification number of the lifeboat that each passenger boarded. The Body variable indicates the identification number of the body of passengers who did not survive. Finally, the Home.dest variable provides information about the home or destination of each passenger.

The dataset is incomplete as some of the fields were missing for some samples and were marked empty, particularly in the fields of age, fare, cabin, and port. However, all the sample points have information on the gender and passenger class. To standardize the data, we filled in the missing values or just dropped the variables that we deemed are too hard to process. It will be discussed in the feature engineering section later.

3 Exploratory Data Analysis

When embarking on a machine learning project, exploratory data analysis (EDA) is a crucial step to understand the dataset we are working with. EDA involves examining and summarizing the main characteristics of the data in order to gain insights and identify patterns. In this section, we will discuss how we conducted EDA on the Titanic dataset. By performing EDA on the dataset, we gained a better understanding of the variables that might be important in predicting whether a passenger survived or not. This knowledge was then used to inform feature selection and engineering, as well as model building and evaluation.

3.1 Univariate Analysis

Univariate analysis involves examining each variable in a dataset separately to determine its distribution. This analysis focuses on the range of values and central tendency of each variable and does not explore relationships between different variables like bivariate and multivariate analysis do. The specific methods used to perform univariate analysis depend on whether the variable is categorical or numerical. For numerical variables, the shape of the distribution, either symmetric or skewed, can be visualized using histogram and density plots. For categorical variables, bar plots can be used to display absolute and proportional frequency distributions. Under-

standing the distribution of feature values is especially important when using machine learning techniques that assume a particular type of distribution, typically Gaussian.

The Survived variable in the dataset is imbalanced as there is a disproportionate representation of survivors and victims in its distribution. Out of a total of 1309 passengers, only 342 passengers survived while a significant 549 passengers perished. This means that 61.8% of the passengers did not survive, while only 38.2% were fortunate enough to survive. Similarly, the Sex variable is imbalanced as the proportion of males and females is not equal. The male gender dominates the Sex variable, accounting for 64.4% of the distribution, while females make up only 35.6%. The Pclass variable also has an imbalanced class distribution as the three categories are not equally represented. Class 3 is the most frequent category with 709 passengers, while Class 2 is the least common. Class 3 makes up over 54% of the Pclass variable, while Class 1 and 2 combined contribute to almost 46%. Finally, the Embarked variable is also imbalanced, with its levels not being equally represented. The majority of passengers embarked from Southampton, while only 123 embarked from Queenstown. This means that almost 70% of the Embarked variable consists of the "S" category, while "C" and "Q" combined contribute to the remaining 30%.

We also counted the frequency of variable Cabin and found that there are a large number of missing values. We also found that the variable is very unorganized. There are 187 kinds of categories and out of 1309 records, there are 1014 missing observations. So we decided to drop this feature in our model because it's very unlikely that we can impute all of the missing values. In addition, variables SibSp and Parch are not balanced as levels of the variables are not equally represented in its distribution. Vast majority of the passengers are without a sibling, spouse, parent, or children accompanying them.

The distrbution of the Fare variable has a high positive skewness, that 653 paid for fare between 5 to 14.9, followed by 191 passengers paid for fare between 25 to 34.9. In contrast, the distribution of the Age variable looks much more normal, although it's still slightly positive skewed.

3.2 Bivariate Analysis

Bivariate analysis is a crucial part of data analysis as it explores the relationship between two variables. This analysis examines the correlation or association between predictor and target variables. It is conducted for any combination of categorical and numerical variables.

The examination of Sex versus Survival reveals that female passengers had a higher likelihood of surviving than male passengers. While 72.7% of female passengers survived, over 80% of male passengers did not. In addition, we analyzed PClass versus Survival and made an interesting observation. Although passengers in Class 1 had the highest number of survivors compared to the other two classes, the percentage data told a different story. Passengers in Class 1 had a survival rate of more than 61.9%, while those in Class 2 had a 43% chance of survival. Astonishingly, Class 3 passengers had only a 25.5% chance of survival.

Our previous analysis showed that a significant number of passengers did not have any siblings or spouses, and then we found that over 65% of them did not survive. However, we observed that passengers were more likely to survive if they had a greater number of siblings and spouses. This trend was also evident in the analysis of Parch versus Survival; the number of parents or children aboard affected the passengers' chance of survival. Passengers who had a higher number of parents or children were more likely to survive.

4 Feature Engineering

We made an interesting observation during our analysis: a passenger's socioeconomic status could be a strong indicator of their chances of survival. Although the Name variable initially seemed unimportant, we noticed that it contained titles such as Mr, Mrs, and Master that provided valuable information

about the passenger's sex, age, and profession, which in turn could influence their survival. We extracted these titles and classified them into broader categories to simplify the analysis.

Additionally, our bivariate analysis revealed that family size had an impact on passengers' survival rates. To address this, we created a new variable called pFamily, which was the sum of Parch and Sibsp plus 1 and indicated each passenger's family size. However, we also noticed that several family sizes had very low frequencies, so we combined them into four categories: single, small, medium, and large. This helped to reduce complexity and streamline the analysis.

Imputing missing values is a crucial step in feature engineering to avoid pitfalls caused by missing data. One way to do this is by using mean, median, or mode imputation depending on the variable type (categorical or numerical) and its distribution. For example, mode-imputation is used for categorical variables, while mean-imputation is used for numerical variables with symmetric or normal distributions, and median-imputation is recommended for skewed distributions with outliers, such as Fare. However, a potential drawback of using mean, median, or mode imputation is that it can introduce bias if the amount of missing data is significant, such as in the case of Age. In this scenario, a better approach is to group the data by variables that have no missing values and compute the median age for each subset to impute missing values. Another alternative is to build a linear regression model that predicts missing values of Age based on features that have no missing values. These methods are likely to result in better accuracy with less bias, except when missing values are expected have high variance.

We utilized different imputation methods for different variables with missing values. Since there were only a few missing values in Fare and Embarked, as Fare only had 1 and Embarked only had 2, we chose to impute the missing values with the median and mode, respectively, as we anticipated that this approach would not introduce any bias. For Age, we opted to group the data by variables that were highly correlated with it and calculate the median values of each group to impute the missing values. We first created a boxplot to visualize the correlation between Age and other variables. We observed that Sex and Embarked did not seem to have a significant impact on Age, whereas Pclass, pName, and pFamily did. We then generated a heatmap to examine the correlation among these variables and discovered that pName and Pclass were more strongly correlated with Age than with other variables. Consequently, we grouped Age by pName and Pclass and imputed the medians of the subgroups.

Next, we performed binning on Age and Fare to convert them into categorical variables. This was done to prevent overfitting, a common issue with tree-based models such as the Random Forest Classifier, which we planned to use. To bin Age, we followed the age designations of the American Medical Association and grouped it into categories of infant, child, teenager, young adult, adult, and aged. As for Fare, we categorized it into groups of low, medium, high, and very high.

We proceeded to drop features that were redundant, irrelevant, or could potentially lead to overfitting. To avoid increasing the complexity of the model unnecessarily, we removed Name, Sibsp, Parch, Age, and Fare, as they already had processed versions available. Additionally, we discarded Cabin because it had too many missing values, and we eliminated Ticket, Boat, and Body because they were merely identification numbers and did not contain relevant information for our machine learning models. We then proceeded with encoding our variables. We opted for one hot encoding over label encoding to avoid any potential weight bias that may occur in the algorithm when using label encoding.

To further reduce complexity, we conducted a chi-square test to determine the relationship between our encoded features and target columns. We sorted the features by their p-values, where the null hypothesis stated that there is no relationship between the feature and the target, while the alternative hypothesis stated that there was a relationship. If the p-value is significant (less

than 0.05), we can reject the null hypothesis and conclude that the findings support the alternate hypothesis. As a result, we extracted only the encoded variables with a p-value less than 0.05. We then split the data using train_test_split function from sklearn [2]. At this point, the dataset was fully processed and ready to be used for model training.

5 Random Forest

In solving complex problems, seeking the insights of multiple experts from diverse backgrounds is often necessary to arrive at a sound decision. Each expert brings their expertise and experience to provide their opinion, which is then collectively voted on to determine the final decision.

Similarly, in the field of machine learning, a random forest classifier employs multiple decision trees using different random subsets of the data and features. Each decision tree represents an expert, offering its own classification opinion. Predictions are made by computing the prediction for each decision tree and selecting the most popular result in classification problems. For regression problems, an averaging technique is utilized to produce the prediction.

Random Forest Classifier presents several advantages over other algorithms, such as decision trees or logistic regression. Firstly, it reduces the risk of overfitting by combining the outputs of multiple decision trees. Secondly, it can effectively manage large datasets with high dimensionality, making it highly scalable. Lastly, it can handle both categorical and numerical variables.

We observed that the distribution of the target variable, survived, was imbalanced and needed to be addressed when using the Random Forest Classifier from sklearn. The class_weight parameter in the Random Forest Classifier is specifically designed to handle imbalanced class distributions in the dataset. By default, the classifier assumes that each class has equal weight, which is not always true in real-world datasets, including ours. To overcome this, we utilized the class_weight parameter to assign different weights to different classes. This approach directed the algorithm to pay more attention to the minority class during training, ultimately improving the model's ability to predict the minority class. The weight calculation was determined by total observations/(n_classes*observations in class).

Once we had assigned the class weight, we proceeded with the application of the Random Forest model. We instantiated the model and fitted it to our training data by passing both the features and target variable. Subsequently, to evaluate the model's accuracy, we compared its predictions to the actual values in the test set. We used several metrics to assess the model's performance, including accuracy, precision score, recall score, and specificity score.

Accuracy score is the simplest and most widely used metric for assessing model performance. It measures the proportion of correct predictions made by the model. Precision score represents the ratio of true positives to total predicted positives, providing insights into the model's ability to identify positive instances correctly. On the other hand, recall score calculates the ratio of true positives to total actual positives, indicating how well the model captures actual positives. Finally, specificity score measures the ratio of true negatives to total actual negatives, offering a measure of the model's ability to identify negative instances correctly.

To further enhance the accuracy of the classifier, we utilized Scikit-Learn's GridSearchCV function to fine-tune its hyperparameters. With GridSearchCV, we could specify the hyperparameters we wanted to optimize, as well as the range of values to explore for each parameter, in a dictionary called rfParams. By doing so, we could automate the hyperparameter tuning process, which is essentially the same as fitting a Random Forest model, but with a focus on finding the optimal hyperparameter values.

Before hyperparameter tuning, we tested our model with both training and testing set and received the following outcome: the training set prediction receives accuracy score of 0.8439, precision score or 0.7976, recall score of 0.7765, and specificity score of

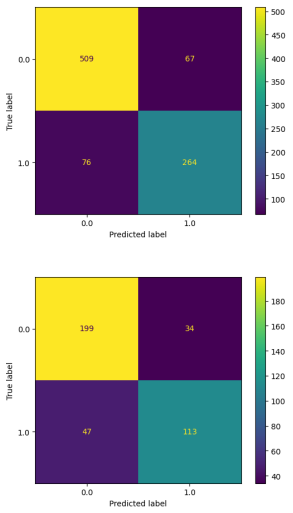

Figure 1: (a) Prediction confusion matrix on the training set before hyperparameter tuning (b) Prediction confusion matrix on the testing set before hyperparameter tuning

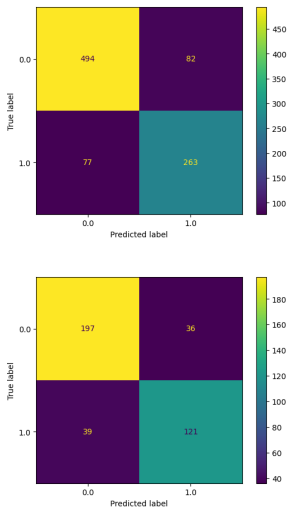

Figure 2: (a) Prediction confusion matrix on the training set after hyperparameter tuning (b) Prediction confusion matrix on the testing set after hyperparameter tuning

0.8837; the testing set prediction receives accuracy score of 0.7939, precision score of 0.7687, recall score of 0.7063, specificity score of 0.8541.

After hyperparameter tuning, we tested our model with both training and testing set and received the following outcome: the training set prediction receives accuracy score of 0.8264, precision score or 0.7623, recall score of 0.7735, and specificity score of 0.8576; the testing set prediction receives accuracy score of 0.8092, precision score of 0.7707, recall score of 0.7563, specificity score of 0.8455.

Therefore, it seems that base hyperparameters are actually producing better results than the GridSearchCV one.

The total run time of the entire program, including the EDA part and hyperparameter tuning part which takes a vast majority of run time, is 87.36 seconds.

6 Support Vector Machhine

SVMs have demonstrated effective generalization performance across diverse problem domains such as handwritten character recognition, face detection, pedestrian detection, and text categorization. Nonetheless, one drawback of SVMs is that their training algorithms can be sluggish, particularly for large-scale problems. Additionally, the training algorithms for SVMs can be complex, subtle, and challenging for average engineers to implement. To address this, we have employed a new SVM learning algorithm known as Sequential Minimal Optimization (SMO) [3], which is generally faster than the standard SVM training algorithm.

The simplest form of an SVM is a hyperplane that separates a set of positive examples from a set of negative examples while maximizing the margin. In the linear case, the margin is determined by the distance of the hyperplane from the nearest positive and negative examples. The formula for the output is:

$$f(x) = w^T x + b$$

Given that our problem involves binary classification, we will predict y=1 if $f(x) \geq 0$ and y=-1 if $f(x) < 0$. By lookinig at the dual problem, we observe that the above equation can also be represented using inner products as

$$f(x) = \sum_{i=1}^{m} \alpha_i y_i x_i x + b$$

where we can substitute a kernel $K(x_i, x)$ in place of $x_i x$.

The SMO algorithm provides an effective means of addressing the dual problem of the optimization problem for support vector ma-

chines. In particular, our goal is to solve:

$$max_\alpha \quad \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

subject to $0 \leq \alpha_i \leq C$, i=1,...,m

$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

And KKT conditions are used to verify whether the optimization process has converged to the optimal point. In particular, KKT conditions are:

$$\alpha_i = 0 \rightarrow y_i(w^T x_i + b) \geq 1$$

$$\alpha_i = C \rightarrow y_i(w^T x_i + b) \leq 1$$

$$0 < \alpha < C \rightarrow y_i(w^T x_i + b) = 1$$

To state it differently, any α_i's that fulfill these properties for all i will represent an optimal solution to the optimization problem mentioned previously. The SMO algorithm continues to iterate until all of these conditions are met (within a specified tolerance level), thus ensuring convergence.

6.1 SMO algorithm

In summary, the SMO algorithm selects two α parameters, α_i and α_j, and jointly optimizes the objective value for both of these parameters. It achieves this by identifying an appropriate pair of α_i and α_j that do not satisfy the KKT conditions and then carrying out an iterative process to adjust these values. During each iteration, the algorithm selects a suitable variable to optimize, and then applies optimization techniques to adjust its corresponding α value. Once the optimal values of α_i and α_j have been obtained, the algorithm proceeds to adjust the value of the bias parameter, b, based on these new α values. This adjustment ensures that the classification hyperplane is shifted in the appropriate direction. The process of selecting and optimizing α parameters, followed by adjusting the bias parameter, is repeated until the values of α converge to a solution that satisfies the KKT conditions within a certain tolerance level. This guarantees that the algorithm has found an optimal solution to the support vector machine optimization problem.

To explain it with more detail, Once the Lagrange multipliers α_i and α_j have been selected for optimization, the first step is to calculate constraints on their possible values. This involves determining lower and upper bounds, denoted as L and H, respectively, such that the constraint $0 \leq \alpha_j \leq C$ is satisfied. The bounds L and H ensure that α_j remains within the allowed range while optimizing the objective function. Specifically, L represents the lower bound on α_j, while H represents the upper bound. These bounds must satisfy the condition $L \leq \alpha_j \leq H$, which guarantees that the constraint $0 \leq \alpha_j \leq C$ is not violated during the optimization process. Once the bounds have been determined, they are used to solve the constrained maximization problem and obtain the optimal values of α_i and α_j. The bounds can be found as the following:

If $y_i \neq y_j, L = max(0, \alpha_j - alpha_i), H = min(C, C + \alpha_j - \alpha_i)$

If $y_i = y_j, L = max(0, \alpha_i + alpha_j - C), H = min(C, \alpha_i + \alpha_j)$

The next step is to maximize the objective function by finding the optimal value of α_j. However, it is possible for this value to exceed the pre-determined bounds, L and H. In such cases, we need to ensure that α_j lies within the allowed range by clipping its value to the nearest bound.

$$\alpha_j^{new} = \alpha_j^{old} + \frac{y_j(E_i - E_j)}{\eta}$$

where $E_i = f(x^i) - y_i$

$$\eta = K(x_i, x_i) + K(x_j, x_j) - 2K(x_i, x_j)$$

We clip α_j within the range of L and H:

$$\alpha_j = \begin{cases} \alpha_j > H \rightarrow H \\ L \leq \alpha_j \leq H \rightarrow \alpha_j \\ \alpha_j < L \rightarrow L \end{cases}$$

After we found α_j, we can thus find α_i by computing the following:

$$\alpha_i^{new} = \alpha_i^{old} + y_i y_j (\alpha_j^{old} - \alpha_j^{new})$$

Once the Lagrange multipliers, which are α_i and α_j, have been optimized, the threshold parameter, denoted as b, is selected to ensure that the KKT conditions are satisfied for the i^{th} and j^{th} examples. If α_i is found to be non-bound after optimization, we can obtain a valid threshold parameter, denoted as b_1 or b_2, which ensures that the SVM outputs y_i when the input is x_i.
$b_1 = b - E_i - y_i(\alpha_i - \alpha_i^{old})K(x_i, x_i) - y_j(\alpha_j - \alpha_j^{old})K(x_i, x_j)$
$b_2 = b - E_j - y_i(\alpha_i - \alpha_i^{old})K(x_i, x_j) - y_j(\alpha_j - \alpha_j^{old})K(x_j, x_j)$

If both optimized Lagrange multipliers are found to be at the bounds, then any threshold value between b_1 and b_2 that satisfies the KKT conditions is valid. In this case, we can set the threshold parameter b to the average of b_1 and b_2, in other words $b^{new} = \frac{b_1+b_2}{2}$

6.2 Code explanation

Unlike the RF model algorithm, this model was structured in object-oriented programming style. It is largely due to the fact that there are too many parameters and computations that are required to be used repeatedly. Our algorithm has multiple methods:

__init__(self, C, kernel, epsilon) is the constructor of the SVM class which initializes the SVM parameters. C is the regularization parameter, kernel is the kernel function to be used for transforming data, and epsilon is the tolerance for stopping criterion.

kernel_function(self, x1, x2) omputes the kernel function value between two input vectors x1 and x2, using the kernel function that was specified during the initialization of the class. The available kernel functions are linear, Gaussian, polynomial, laplacian, and exponential. For the Gaussian kernel function, the default value of parameter σ is 1. For the polynomial kernel function, the default power of the exponential is set to 3, while for the laplacian kernel function, the default value of parameter σ is set to 2, and the default value of parameter γ is set to 5.

fit(self, train_x, train_y, max_iter = 1000) trains the SVM on the input training data train_x and corresponding target labels train_y. It uses SMO algorithm to optimize the SVM parameters. And max_iter specifies the maximum number of iterations allowed for convergence.

SMO_get_alpha(self) returns a pair of alpha values to be optimized by SMO algorithm based on the KKT conditions.

decision_function(self, x) computes the decision function for the SVM for the input vector x.

error(self, index) computes the error between the predicted and actual output of the SVM for the training example at index "index".

choose_another_alpha(self, index) selects another alpha value to be optimized together with the alpha value at "index".

SMO_train(self, index1, index2) performs SMO optimization by updating two alpha values given by index1 and index2. It returns a Boolean value indicating whether convergence has been reached.

compute_alpha2(self, index1, index2, x1, x2, y1, y2, old_alpha) computes the updated value of alpha2 for SMO optimization.

compute_LH(self, y1, y2, alpha, index1, index2) computes the bounds for alpha2 based on alpha1 and target labels.

clip_alpha(self, alpha, L, H) clips the updated value of alpha2 to be within the bounds L and H.

compute_b(self, x1, x2, y1, y2, alpha1, alpha2, old_alpha, index1, index2) computes the bias value for the SVM which will be used in **compute_bias(self, alpha1, alpha2, y1, y2, b1, b2)**.

compute_bias(self, alpha1, alpha2, y1, y2, b1, b2) computes the bias value for the SVM given the updated values of alpha1, alpha2, and bias values b1 and b2.

transform_one(self, x) handles output from **decision_function(self, x)**, which will return 1 if the output is greater than 0, and -1 otherwise.

predict(self, test_x) does what predict method typically does, which is to apply the machine learning model on the testing set and try to classify each record.

In addition, a large chuck of code in the driver program came from the feature en-

Linear	[1,0.001]	[1,1]	[0.1,0.01]	[0.001,0.1]	[1,10]	[10,0.001]
Accuracy	0.751	0.667	0.654	0.667	0.667	0.593
Precision	0.782	0.561	0.549	0.561	0.561	0
Recall	0.538	0.838	0.838	0.838	0.838	0
Specificity	0.897	0.55	0.528	0.549	0.549	1

Table 1: Linear Kernel performance.

Gaussian	[1,0.001]	[1,1]	[0.1,0.01]	[0.001,0.1]	[1,10]	[10,0.001]
Accuracy	0.593	0.593	0.753	0.448	0.443	0.407
Precision	0	0	0.774	0.424	0.422	0.407
Recall	0	0	0.556	1	1	1
Specificity	1	1	0.888	0.069	0.06	0

Table 2: Gaussian Kernel performance.

Polynomial	[1,0.001]	[1,1]	[0.1,0.01]	[0.001,0.1]	[1,10]	[10,0.001]
Accuracy	0.707	0.73	0.580	0.73	0.73	0.707
Precision	0.8	0.787	0.491	0.787	0.787	0.8
Recall	0.375	0.463	0.856	0.463	0.463	0.375
Specificity	0.936	0.914	0.391	0.914	0.914	0.936

Table 3: Polynomial Kernel performance.

Laplacian	[1,0.001]	[1,1]	[0.1,0.01]	[0.001,0.1]	[1,10]	[10,0.001]
Accuracy	0.593	0.593	0.593	0.407	0.407	0.593
Precision	0	0	0	0.407	0.407	0
Recall	0	0	0	1	1	0
Specificity	1	1	1	0	0	1

Table 4: Laplacian Kernel performance.

gineering part of the Random Forest model program to ensure that both models uses the same dataset.

In the driver program, we also included algorithm to plot hyperplane. We utilized PCA function from sklearn.decomposition to transform the feature space into 2-D so that we can plot the hyperplane and the features on a 2-D graph. It first creates a grid of points using np.linspace() within the range of -4 to 4 for both x and y axes. For each point in the grid, it computes the SVM's decision function using svm.decision_function() and checks if it is close to 0. In this case, we check if it falls within the range of -0.01 to 0.01. If it does, then the point is added to a list of points called points.

After creating the points list, the code plots the original data points using plt.scatter() with different colors based on their class labels y. It then plots the points on top of the data points to visualize the hyperplane.

6.3 Result

We tried 5 kernel functions, linear, Gaussian, polynomial, laplacian, and exponential. We used the same 6 combinations of C and ϵ for each kernel functions to test their performance, and the combinations are [1, 0.001], [1, 1], [0.1, 0.01], [0.001, 0.1], [1, 10], [10, 0.01]. There are 5 distinct values for C and ϵ. The results of these kernel functions are recorded in the tables.

Both the Laplacian Kernel and Exponential Kernel are the worst performing kernels, as they have not accurately predicted any records. For both of these kernels, all attempted parameters resulted in either a recall score of 0 and specificity score of 1 or a recall score of 1 and specificity score of 0. A recall score of 0 with a specificity score of 1 suggests that the model is being excessively cautious in its predictions, correctly identifying all negative instances but failing to identify any positive ones. Conversely, a recall score of 1 with a specificity score of 0 suggests that the model is being too aggressive in its predictions, correctly identifying all positive instances but failing to identify any negative ones.

The Gaussian Kernel is the next worst performing kernel with only one combination of parameters achieving good performance, having an accuracy of 0.753, precision of 0.774, recall of 0.556, and specificity of 0.888. Out of the six parameter settings, two resulted in a recall of 0 and specificity of 1, and one had a recall of 1 and specificity of 0, while the remaining two had a recall of 1 and specificity above 0, but still extremely low.

In contrast, the linear kernel is the second-best performing kernel, with only one parameter setting resulting in a recall of 0 and specificity of 1. Setting [1,0.001] performed the best with respect to accuracy, precision, and specificity.

The best performing kernel is the polynomial kernel. Every setting has gotten good scores. All the settings except the third one got similar performance with respect to accu-

Exponential	[1,0.01]	[1,1]	[0.1,0.01]	[0.001,0.1]	[1,10]	[10,0.001]
Accuracy	0.593	0.593	0.593	0.407	0.407	0.407
Precision	0	0	0	0.407	0.407	0.407
Recall	0	0	0	1	1	1
Specificity	1	1	1	0	0	0

Table 5: Exponential Kernel performance.

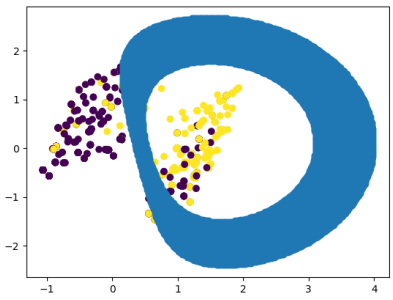

Figure 3: Plot of hyperplane for Gaussian Kernel

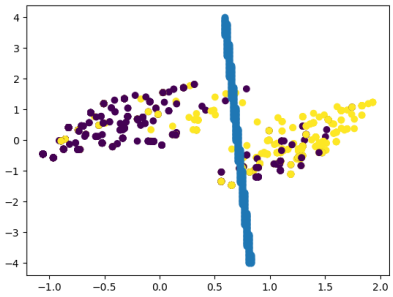

Figure 4: Plot of hyperplane for Linear Kernel

racy, precision, recall, and specificity. Setting 3, on the other hand, has lower accuracy and precision scores compared to the other settings, indicating that it is making more incorrect predictions. However, its recall score is high at 0.856, suggesting that it is able to capture a higher proportion of actual positive instances compared to the other models. Its specificity score is low at 0.391, indicating that it is not able to identify as many true negatives out of all actual negatives. It's hard to conclude which one is the best fit for our dataset, but setting 2, 4, and 5 all have higher accuracy and much better recall compared to the other models that have similar performance.

There are a few things we need to be aware of or conclude. As C increases, the ability to fit non-linear data increases, but it is more prone to overfitting. The larger the parameter σ of the Gaussian kernel, the smoother the function, and the worse its ability to fit non-linear data, and the less sensitive it is to noise. If a kernel function is used, feature

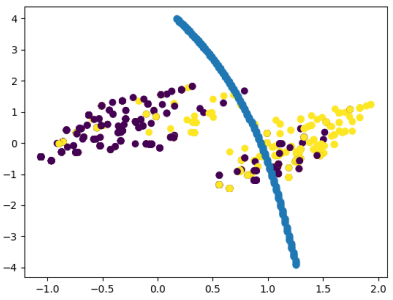

Figure 5: Plot of hyperplane for Polynomial Kernel

scaling, or normalization must be performed on the features. If the training set is too small but the number of features is large, the training data may not be sufficient to fit complex non-linear models. In this case, only the linear kernel can be used and the Gaussian kernel cannot be used.

In the case of running a Linear Kernel function, the total run time is 9.27 seconds if we also choose to plot the hyperplane, the run time is 0.42 seconds if we don't plot the hyperplane.

Conclusions

We utilized both Random Forest classifier and Support Vector Machine to predict passenger survival on Titanic. To enhance our project, we could focus on improving feature engineering since hyperparameter tuning did not significantly enhance our RF model scores. As the dataset is not particularly large, we should consider normalization and improve our binning of continuous variables to minimize information loss. Regarding the SVM program, plotting the hyperplane was the main time-consuming process, running in $O(n^2)$, which decreased efficiency. Given that we use Kernel functions, normalization is crucial in this scenario. We must also experiment with more hyperparameters to achieve better scores. Exploring other features, methods, or machine learning algorithms would be fascinating to continue this analysis.

Credits

We used scikit-learn, a popular Python library for machine learning, to implement our models. We also used Sequential Minimal Optimization, an algorithm developed by John C. Platt of Microsoft, used for training Support Vector Machines. We greatly appreciate all their works.

References

[1] The original Titanic dataset, describing the survival status of individual passengers on the Titanic. The titanic data does not contain information from the crew, but it does contain actual ages of half of the passengers. The principal source for data about Titanic passengers is the Encyclopedia Titanica. The datasets used here were begun by a variety of researchers. One of the original sources is Eaton & Haas (1994) Titanic: Triumph and Tragedy, Patrick Stephens Ltd, which includes a passenger list created by many researchers and edited by Michael A. Findlay.

Thomas Cason of UVa has greatly updated and improved the titanic data frame using the Encyclopedia Titanica and created the dataset here. Some duplicate passengers have been dropped, many errors corrected, many missing ages filled in, and new variables created.

For more information about how this dataset was constructed: http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3info.txt

[2] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[3] Platt, John. (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Advances in Kernel Methods-Support Vector Learning. 208.

A Appendix: Graphs

A.1 Univariate Analysis

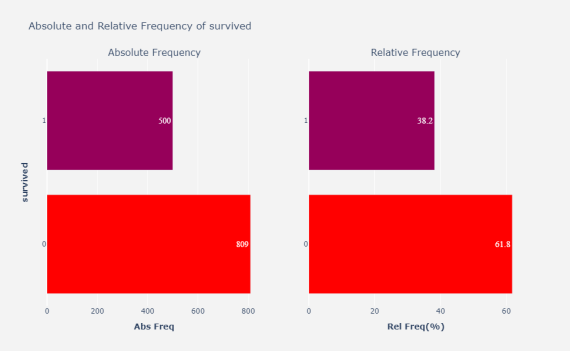

Figure 6: Plot of frequency of Survived variable

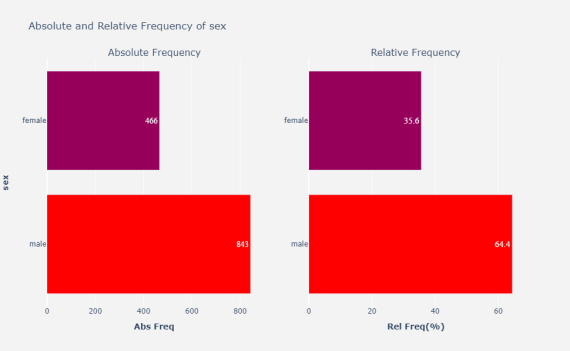

Figure 7: Plot of frequency of Sex variable

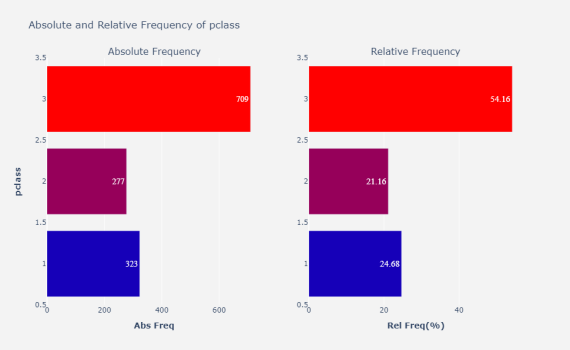

Figure 8: Plot of frequency of Pclass variable

A.2 Bivariate Analysis

A.3 Feature Engineering

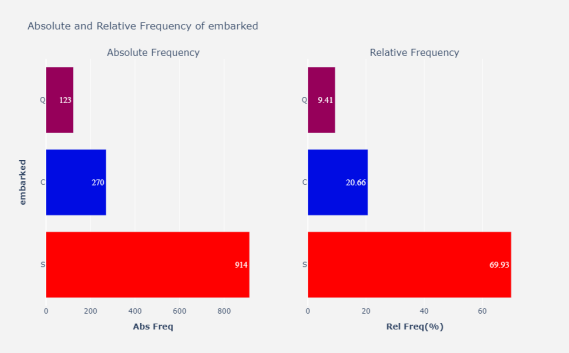

Figure 9: Plot of frequency of Embarked variable

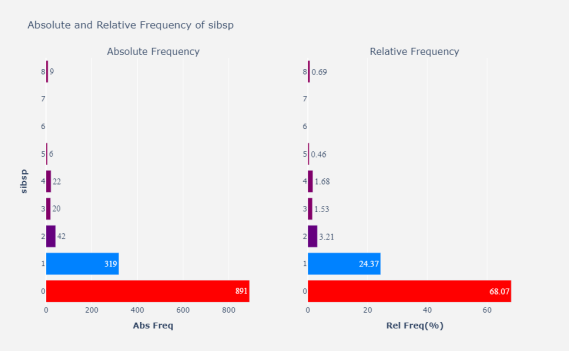

Figure 10: Plot of frequency of SibSp variable

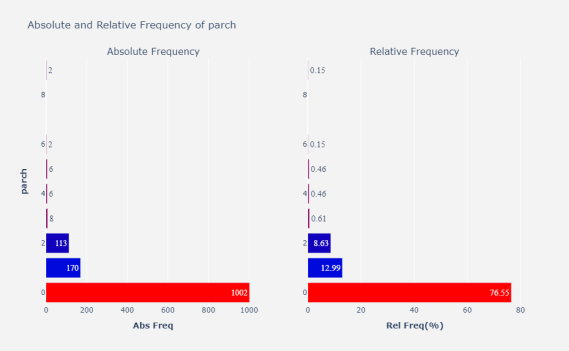

Figure 11: Plot of frequency of Parch variable

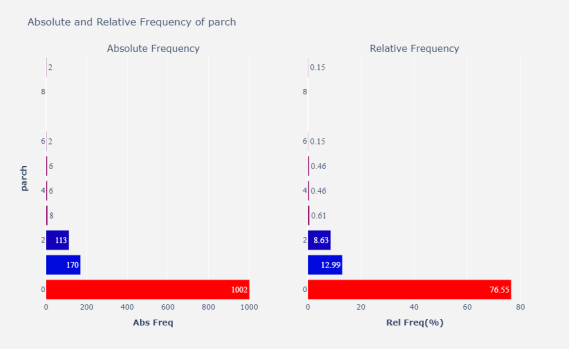

Figure 12: Plot of frequency of Parch variable

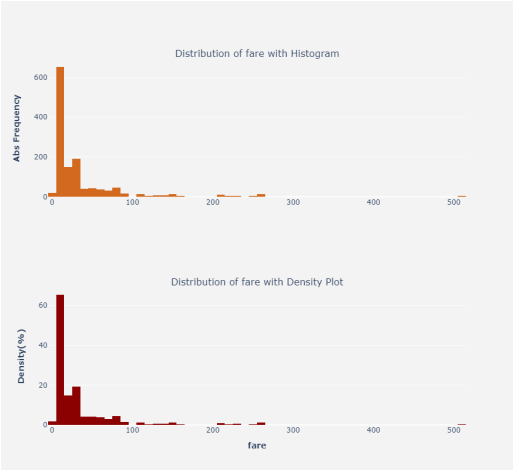

Figure 13: Distribution of Fare variable

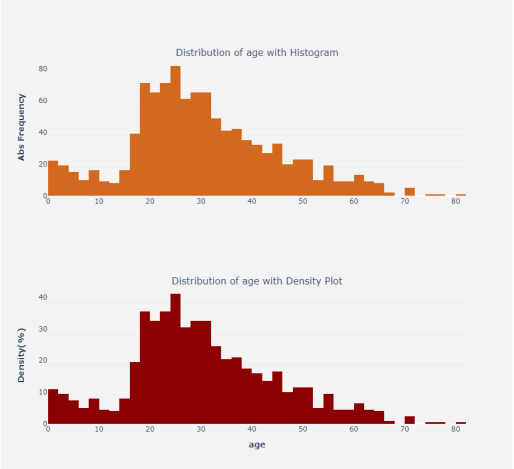

Figure 14: Distribution of Age variable

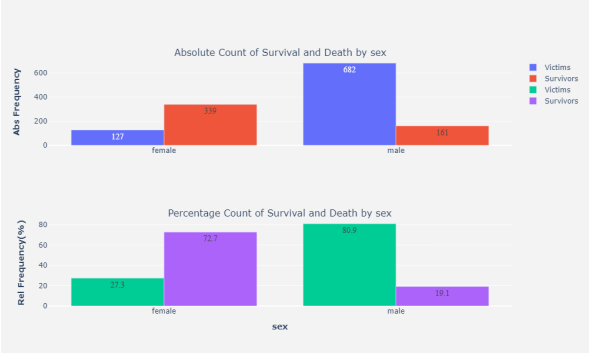

Figure 15: Sex vs. Survived

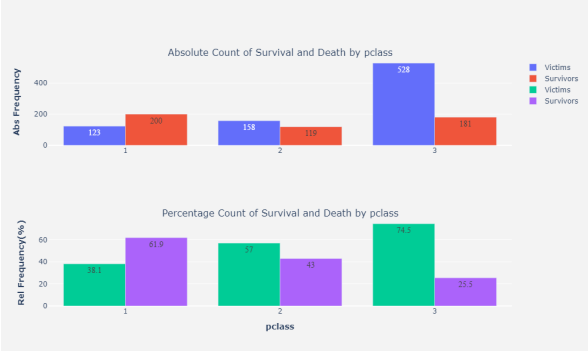

Figure 16: Pclass vs. Survived

Figure 17: SibSp vs. Survived

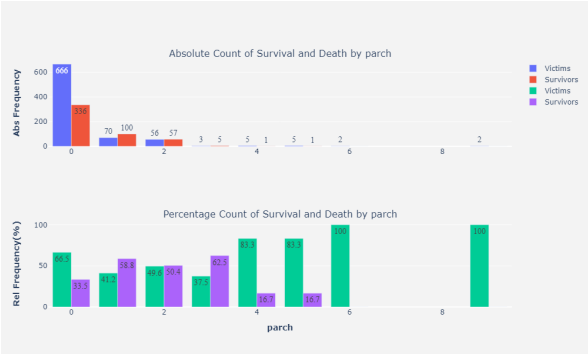

Figure 18: Parch vs. Survived

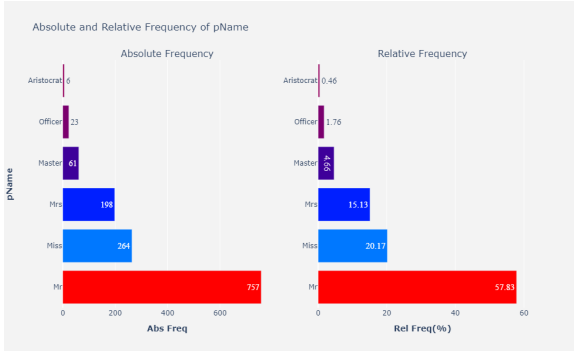

Figure 19: Frequency of pName

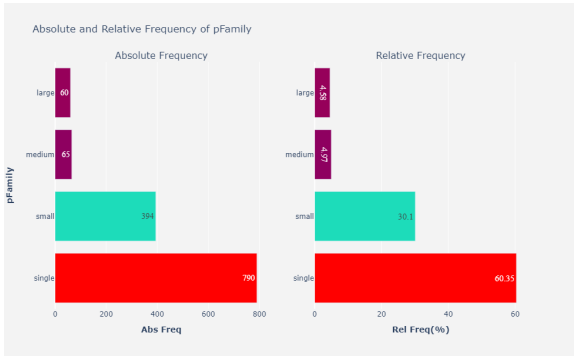

Figure 20: Frequency of pFamily

Figure 21: Number of missing values across the dataset

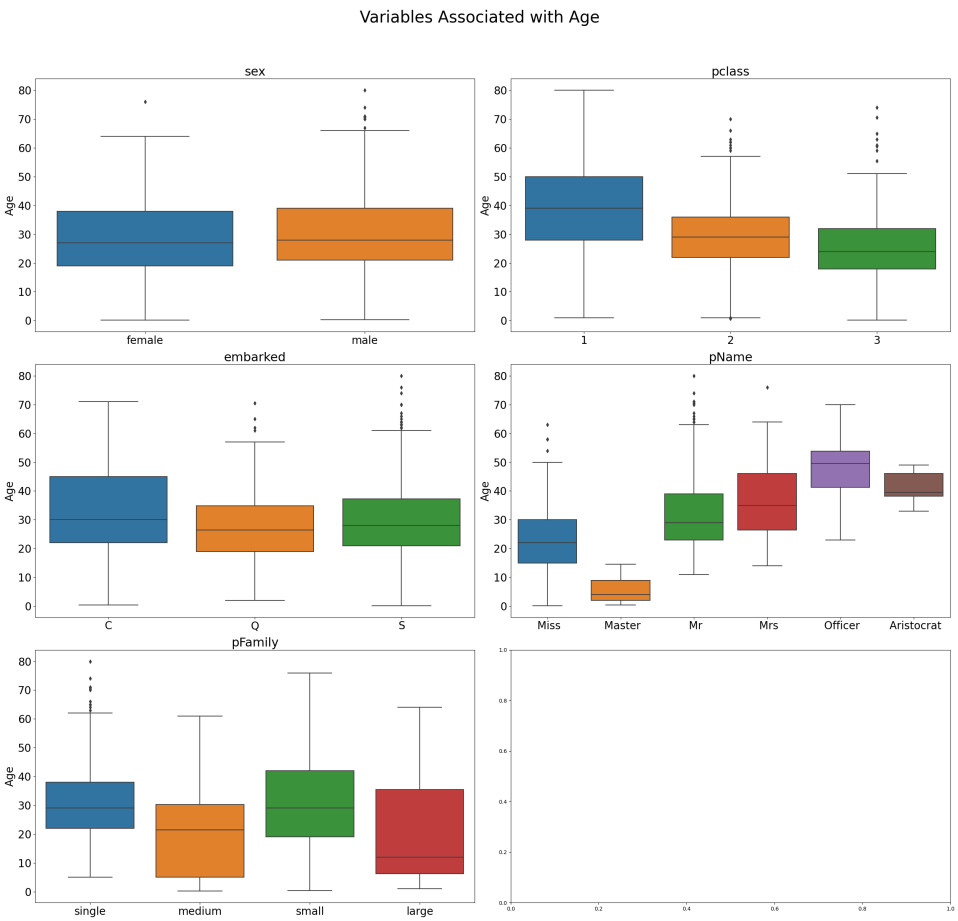

Figure 22: Variables that may associate with Age

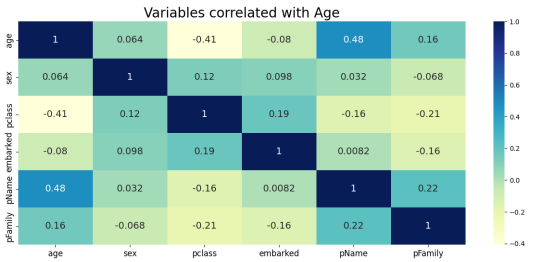

Figure 23: Correlations between Age and variables

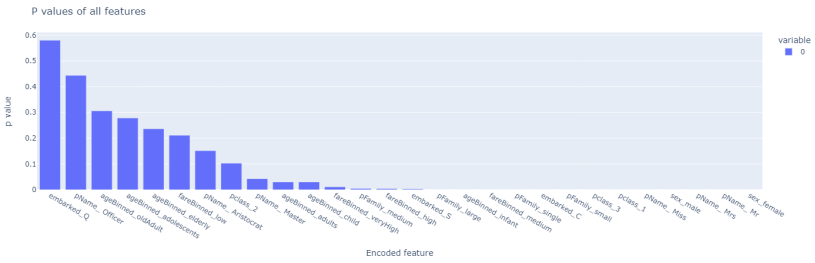

Figure 24: P-values in feature selection