Different Machine Learning Approaches on Kaggle's Titanic Dataset:

Joshua Bornstein

Introduction

In this project, the primary objective was to apply machine learning algorithms to the Kaggle titanic dataset, attempting to predict whether or not a passenger survived based on their name, ticket id, ticket fare, passenger class, sex, age, number parents/children on board with them, number of siblings/spouses on board with them, their cabin, and place of embarkment, while using at least three distinct categories of feature learning approaches: Fix-shape universal approximators, Neural network-based universal approximators, and Tree-based approaches. As well as using those, I also attempted to do feature engineering and trained a number of dummy models using different rules. The metric I use to compare these models was overall accuracy on the validation set.

Methodolgy

I used a number of libraries including pandas, numpy, sklearn and tensorflow for this project. I downloaded the data into an iPython notebook. First, I did exploratory data analysis, then I implements 3 different preprocessers, then I trained linear, kernel, neural network, decision tree, and random forest models on the preprocessed data sets that were compatible. Then I tuned my hyperparameters for each model as well as preprocessing.

Dataset Exploration and Pre-processing

First I loaded in the Kaggle dataset and conductied exploratory data analysis. I made plots of coorelations between different features and survival rates, as well as between features and survival rates on specific subsets of the data. I additionally conducted statistical analysis both on whether age recorded, and cabin recorded were predictors of survival and found that they both were. I also extracted the numberical data points and plotted them according to the top 2 principal components, which did not show any discernable separabilities.

For the sake of shortening my code submission I removed significant parts of my data exploration including many interesting results about single women, and single men versus women and men with families.

I employed 3 different preprocessing steps: A, B, and C

Preprocessing step B uses one-hot encoding and is never normalized. C is also never normalized, but the the values of A were normalized when training fixed-shape approximators and neural networks. Normalization was not needed for convergence linearly.

For A and B: I decided to remove name, ticket number, and passenger id from the data sets. I explain my reasoning for this further in the iPython notebook.

The testing data differents slightly from the training data in that there is a single passaneger without a fare. I replaced it with the median fare.

A: I encoded cabin as a binary variable indicating whether or not cabin was recorded. I replaced the 2 unknown embarked entries with the mode embarked. I decided to break age into a categorical group so I could better deal with the unkown group. I placed the unkown age group as the oldest age group. And left them with linearly ordered labels. I hyperparameter tuned my choice of age group splits.

B: I encoded cabin as a binary variable indicating whether or not cabin was recorded. I replaced the 2 unknown embarked entries with the mode embarked. I decided to break age and fare into categorical groups. I used one-hot encoding to then turn my matrix into a sparse matrix with 34 columns.

C: I only removed passenger id for this pre-processing step. I used a preprocessing step from a kaggle titanic demo. In this one, I split the names and ticket ids by spaces and strip the commas. I tokenize the names, and I turn the ticket ids into 2 separate features, ticket_number and This preprocessing step was solely used for tfdf models.

After looking at all the data, I hypothesize that a decision tree will be the model due the fact that most of the variables are categorical and that there are a small enough number of them. Unfortunately, the sample size is not very large, compared to the number of combined categories.

Fix-shape Universal Approximators (Kernel Methods)

I tried both Gaussian Regression and Kernel Ridge models. Neither of them performed well at all, with the Guassian Process Regression ending with a test accuracy of .63, and Kernel Ridge with a test accuracy of .773

When using preprocessing A, I also standard normalized my data

I also tried using svm and hyperparameter tuned my choice of kernel and constant. My tuned svm model achieved .79 accuracy on my test split and I was unable to test in a kaggle's website because I used all of my tests already and will be submitting the project before I get another chance although the code is still there if you want to do it.

Dummy and Linear Models

To establish a baseline for performance, dummy models were initially trained. Some of the dummy models achieved test accuracies as high as .82 and validation acuracies up to .79

I also trained a few models which kept a number of bins according to some subset of the categorical features and made predictions based of whether the bins of a test data point had more survived or dead people in it.

Then I trained Logistic Regression models using sklearn using both preprocessing A and B and was able to achieve accuracy up to .81 on my test data and .76 on kaggle's validation set.

I also trained individual linear models per each feature to get a sense of how correlated each feature was and trained models, and tried splitting the data by gender and training 2 models and combining which achived a test accuracy of .81

Tree-based Approaches

Decision tree models were implemented and their structures visualized to gain a deeper understanding of the decision-making processes. The project further explored the power of ensemble learning with Random Forest models, optimizing hyperparameters for enhanced predictive capabilities.

I used sklearn to train a decision tree and random forest on both preprocesses of my data.

I used both sklearn and tensorflow_random_forest structures to learn. I trained on all 3 of my preprocessing steps. I was able to achieve a maximum accuracy of .80 on kaggle's validation set, using preprocessing step C.

I hypertuned sklearn's model for max_depth and criterion

Neural Network-based Universal Approximators (NN)

I used sklearn MLP neural network and on both the A and B preprocessed data. A performed better than B in all cases. I did hyperparameter tuning on alpha, hidden layer sizes, choice of solver and choice of activation function. My best model had .83 testing accuracy and then had .76 validation accuracy on kaggles website.

Results

These were the ideal hyperparamaters for my final models, as well as the cooefficients of my linear model, and best performing dummy model.

Preprocessing A was better in most cases than preprocessing B, the following models used preprocessing A.

Logistic Regression Score: 0.7932960893854749
Logistic Regression Coefficients: [[-5.88069584e-01 -2.47153705e+00 -2.80511206e-01
-2.85008308e-01, -1.03876085e-01 2.28843856e-03 6.06955035e-01 -3.21485710e-01]]
(is length 8 becuase my features are pclass, sex, age, parch, sibsp, cabin, fare, embarked)
Logistic Regression Most Important Features: ['Fare' 'Parch' 'Age' 'SibSp' 'Embarked' 'Pclass' 'Cabin' 'Sex']

My logistic regression scores .81 when using the 34 length data from preproccesor B

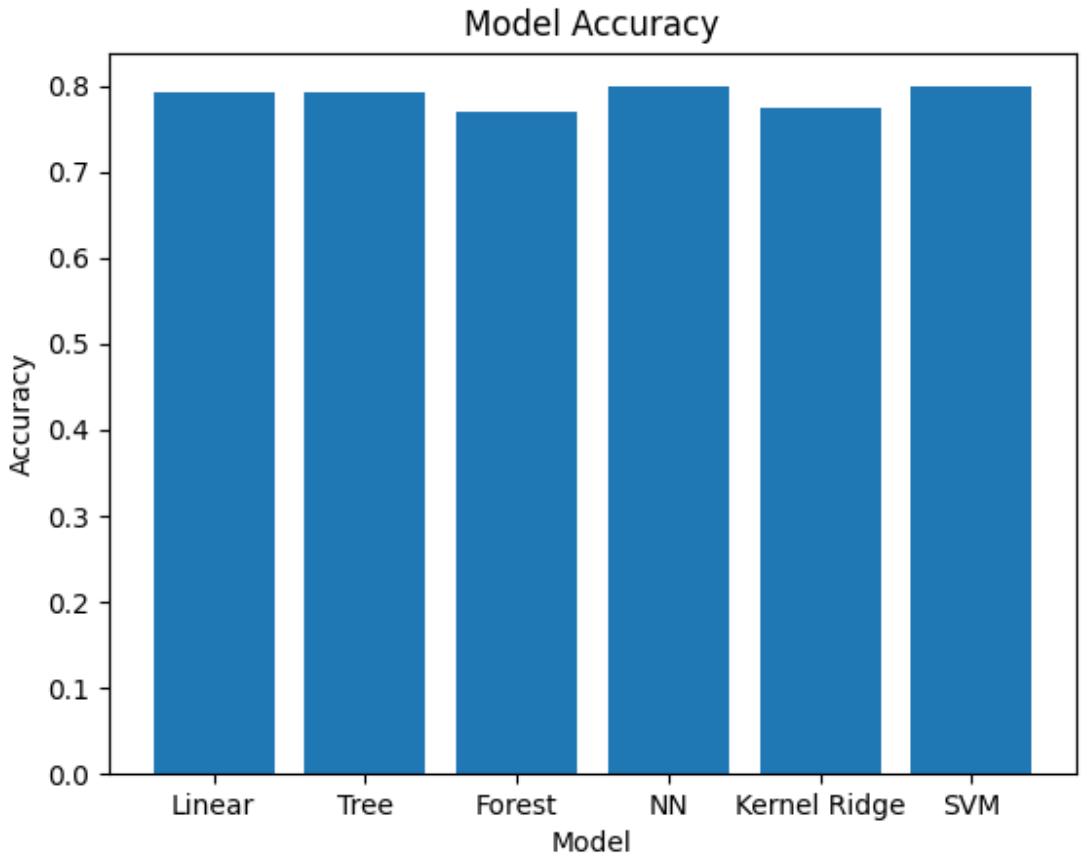
MLP NN Best Params: (hidden layer size = (8, 8, 8), activation function = 'tanh', solver = 'adam', alpha = .1)

SVM Best Parameters: {'C': 1, 'kernel': 'linear'}

Sklearn Decision Tree Best Parameters: {'criterion': 'gini', 'max_depth': 1000}

I have also included a bar graph of model perfomance (on my test split using preproccsing B) but many more graphs and charts are in my code submissinon via iPython notebook (including the

one using A).



Conclusion

While my best model was a tree-based approach as I hypothesized would be best, it hardly outperforms the gender dummy model. My neural network is able to outperform the linear model on my testing data but does not perform better on the validation data which may be an indictor I have overfitted to my test_data via hyperparameter tuning. Since my simpler models performed better on the validation set, it is likely my more complex models are picking up noise as well as patterns.