Jacob Rogers
CS-6350
Shandian Zhe
12/19/22

Final Project Report

Introduction:

Machine learning, the task of training algorithms to computers such that computers can 'learn' to predict on future data, has seen a boom in interest in the past decade and its' influence can be observed throughout social media, online marketing, advertisements, medical screening, polling trends, streaming suggestions (games, movies, music, etc…), 'artificial intelligence', self checkouts at grocery stores and many other central applications. As technology and humanity continue to become so intertwined, being able to effectively build accurate and efficient models, relationships and algorithms is essential for users to gain the most optimized and personalized experience possible.

The purpose of this project is to utilize various machine learning techniques and apply them to a dataset gathered in a 1994 census which details various individuals' information ranging from education, marital status, sex, work class, capital gain/loss, etc… and attempt to use the machine learning models built on this dataset to accurately predict whether an individual will have a yearly salary that is greater than $50,000. The dataset was given as a competitive Kaggle project, which also serves as a means of testing the efficacy of certain machine learning models, but the overarching goal was to obtain the highest accuracy possible on the testing set of the data. Aside from the Kaggle competition, it is clean that an efficient income level prediction model could be valuable in many applications such as an integrated tool used by the IRS, a bank/credit union looking to service a loan to a client, a landlord ensuring a tenant has the available means for renting, or prospective students looking to analyze their future prospects given a roadmap to follow, furthermore, the project allows for a more in depth comparison of some of the most widely used machine learning models based on their use and accuracy given the income dataset.

Data Preprocessing:

The data was initially downloaded from the Kaggle competition link and read into two separate DataFrames in Python using the Pandas library. The categories containing missing data, labeled as "?" were easily identified during the read- in process with Pandas and the consideration of dropping vs keeping the missing labels was considered by analyzing the total percentage of missing data vs the total dataset. It was observed that missing data accounted for approximately 7.4% of the data in both the training and testing data sets. There were to differing opinions on whether to keep or discard these missing values, considering both the train and test sets had approximately the same number of missing values, it seems reasonable to simply discard them, however, replacing the missing values with the majority label for that particular attributes column is another quick and easy fix. Due to this line-of-thought, I had to duplicate the data frames and create versions where the missing data was dropped and the other where it was replaced by majority value. Another factor considered was the distribution of True/False results within the training data set. The distribution of False values was calculated to be approximately 75% of the total data set (with 25% comprising of True predictions)

The next step in preprocessing was to identify the columns that were numerical and those that were categorical, this was simple using Pandas method call: select_dtype, which allowed for the data to be organized by like data-types. Given that numerical data could be used to form a correlation matrix, I decided to examine the correlation matrix for the numerical attributes, the plot in figure 1 below shows the correlation matrix that was generated from the numerical data using Pandas and Seaborn.

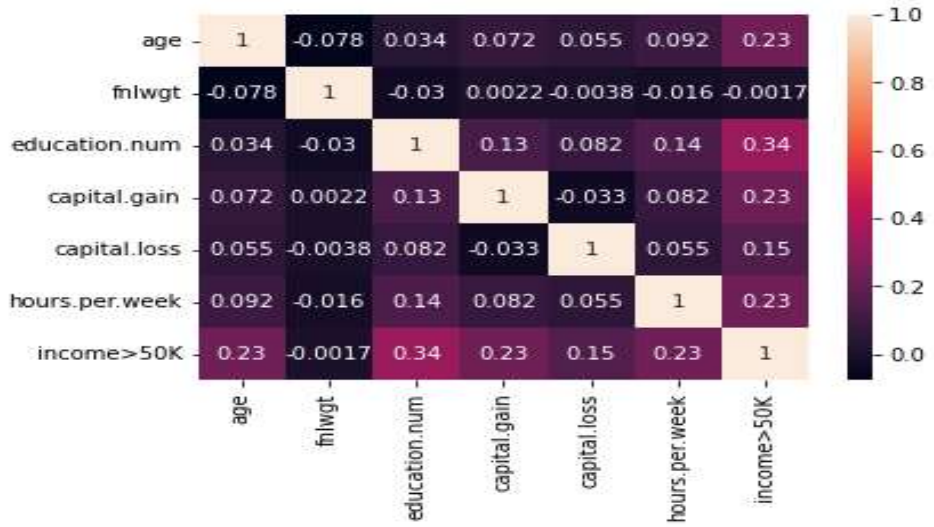

Figure 1: Correlation Matrix for Numerical Attributes in Income Level Prediction Data Set

Using the correlation matrix, I identified that the column attribute ‘fnlwgt’ had a negative correlation to the final prediction outcome, thus hindering any model built using the attribute, because of this, I decided to drop the column entirely from the models training and testing dataset.

The next step in preprocessing was to scale the numerical data and convert the categorical data to numerical types. Scaling the data was simple through the use of sklearn’s preprocessing toolkits. I chose to implement and compare two variations: the MinMax scaler to scale the data evenly between 0 and 1 using the columns maximum and minimum values and StandardScaler. Initially, it seemed like the best option, as it doesn’t affect the overall contribution of a value's weight per column, but simply normalizes the column's value to fall within the expected range, while the standard scaling scales the data to an approximate unit variance of a columns data set.. Encoding the data was a bit trickier and required a lot of reading on StackOverflow, but was ultimately accomplished by using Pandas get_dummies function call and replacing categorical type rows with generated dummy datasets, which were binned such as in a histogram calculation (with 0 to N bins, where N is the number of categorical values the attribute could obtain).

Machine Learning Models And Results:

Four separate machine learning models were applied to the dataset and their parameters (along with further explanation on the logic behind the choices used).

Random Forest was the first model choice, because of the flexibility of classifications produced through decision trees and the reduction in bias and variance from random forest models, additionally, the decision tree schema works well with binary classification. The number of estimators was chosen by incrementing the classifiers and analyzing the training test error until the error was seen towards asymptotic convergence to approximately 99% accuracy, the choice of attribute splitting was chosen to be mean square error, as this was more methodical given the scaling distributed data. The classifier was imported using sklearn decision tree module and the regression variation was used in implementation. Two variations of the random forest model were primarily used in the Kaggle submissions, one which utilized the 'Poisson' means of attribute splitting criterion and the other which used Least Means Square. Each method was used when the data was scaled through MinMax and StandardScaler in order to achieve the highest performance possible. Under the Poisson attribute splitting criterion, the MinMax scaling appeared to suit the data most efficiently, achieving a training accuracy of 95.7%, but the testing accuracy was quite low with a total of 0.7% testing accuracy. The Least Means Square model appeared to fare better when the data was scaled using StandardScaler, achieving a 99% training accuracy and performed as one of the best models on the testing data with a test accuracy of 87.2%.

Stochastic Gradient Descent was the second model implemented, initially as a test comparison against the random forest model, but due to the means of testing accuracy (aside from the training accuracy calculations), which used an AUC (area under the curve) base evaluation model with values falling between [0, 1], higher performance was achieved using regression models versus a binary classification counterpart, as this allowed for a more variable approach to prediction classification (as regressive predictions were forced to fall within [0, 1] without commiting to a specific decision plane). Comparisons were made between using a binary threshold, such as all values greater than 0.5 were labeled 1, otherwise 0, but none of these approaches yielded substantive results, thus the regression predictions were used as the primary model choice. The stochastic gradient descent algorithm was

implemented using the sklearn's module on linear models and the parameters were tuned to 1000 epochs (ensuring convergence and to ensure the regression predictions would yield mostly static results), using hinge loss and an L2 norm for the penalty term. Hinge loss was chosen over 'huber' and 'log-loss' as it had a more consistent approach at dividing true/false predictions, especially with the L2 penalty term, which helped define convergence faster for the training data set. I decided to only test the stochastic gradient descent model (officially) using the hinge loss due to the higher training accuracy, which had a value of 81% under the MinMax scaling and 81.7% under the StandardScaler. The testing accuracy for the stochastic gradient descent was determined to have a testing accuracy of 84.9% (MinMax) and 86.7% for the StandardScaler scaling distribution.

The third model used was Support Vector Machines (SVM), using an 'RBF', Gaussian kernel and was issued until the model reached convergence using a threshold of 10^{-3}, the model was implemented using sklearn's SVM regression model (SVR). Testing the SVM model using the Gaussian kernel and the StandardScaler distribution resulted in a (best of) training accuracy of 84% and test accuracy of approximately 87.0%.

The final model used was that of a neural network. The neural network was based on my implementation of the neural network, created for the Machine Learning course at the U of U, and utilized 3, 4, and 6 layers (using a node width of 20, 30, and 30 respectively), each model running approximately 500 epochs using a sigmoid activation function and mean square loss function. The neural network was modified from a binary classifier to a regressive classifier (by removing the threshold conversion test in the prediction function) and used as the output. The node widths were originally chosen at random and incremented through testing and analyzing the training accuracy score, once the score appeared to reach an equilibrium (not changing significantly), the node width was rounded to the highest point divisible by 10 and chosen as the width. Under the 3 layer neural network with node width of 20, the training accuracy was observed to reach 85.7% with a corresponding test accuracy of 83.5%. Based on the low test accuracy, the layers were increased by one and the node width was adjusted slightly higher, which produced the 4 layer neural network having a training accuracy of 86.2% and test accuracy of 86.8%. In

an attempt to outperform the above results, two more layers were added to the model and the node width was kept constant, producing the 6 layer neural network which obtained a training accuracy of 86.6% and a final test accuracy of 86.4%.

Discussion and Conclusion:

The results above show that the Random Forest model achieved both the highest training and testing accuracy with values of 99% and 87.2% respectively. I initially believed the random forest was suffering from overfitting the data and attempted to limit the depth of the tree, but the results had a significant decrease in testing accuracy (achieving roughly 70% versus the 80%). The second best model used was SVM, which had a close testing accuracy of 87% to the 87.2% random forest model. Stochastic gradient descent and the 4 layer neural network had similar testing accuracies of 86.7% and 86.8% respectively. I believe the limiting factor for the SGD, SVM and NN was the heavy data skew (towards False) in the training data set, which limited the overall capabilities to more accurately form a hyperplane separating the points, but additionally the hyperparameters could have used a much better tuning (through cross validation and further means of testing for convergence). I had high hopes for the neural network computation, which was deemed to converge through analysis of the loss function at each epoch, but the large dataset, high node width and high number of epochs made the model very poor in terms of computational efficiency, whereas the random forest model was significantly faster and had much better coverage over each attribute and possible prediction outcome. All models appeared to approach an asymptotic value of approximately 87% testing accuracy, regardless of their corresponding training accuracy, which indicates that a problem may be within the data pre-processing stage rather than the model testing stage itself. If given more time, I would attempt to approximate the missing data with fractional values for each attribute, apply cross validation for SVM parameters and attempt to restrict the training data to eliminate the heavy data skew and ideally map each model slightly more accurately.

The project code is found at:

https://github.com/Humanfntorch/MachineLearning/tree/main/KaggleCompetition