Final Project Report

Luke Schreiber - CS 5350 - University of Utah

Abstract

In this final competitive project for machine learning, a problem of predicting a person's salary (above or below 50k) from 14 variables was solved using machine learning methods like logistic regression and neural networks. Using a high level method of preprocessing, training and evaluating, the accuracy of predictions increased over time from 50% to 76.6%. Ultimately, this predictor was helpful in better understanding and predicting wealth inequality and income.

Introduction

This final project for Machine Learning (CS 5350) is for the Community Prediction Competition hosted by Professor Zhe on Kaggle. The goal/problem of this project was to predict a person's income level to be above or below 50K, based on a list of 14 demographic and socioeconomic attributes. These include attributes like age, education, occupation, hours per week, etc. The variables include continuous, categorical and integer types. The data for this prediction was extracted by Barry Becker from the Census database in 1994. Solving this problem by creating accurate predictors would help in understanding how these factors contribute to wealth inequality. Understanding how much money a person is likely to make based on these factors can help combat wealth inequality and improve one's financial outcome. This project not only helps us practice our machine learning skills on a dataset, but allows us to gain real world insights into how income or other socioeconomic variables can be predicted.

Using machine learning techniques help more accurately find which predictors correlate best with certain outcomes. Using standard methods of analysis for such complicated multivariate data can only get so far in building a fine tuned prediction model. When there are thousands of entries and 14 variables, the ability for a human to sift through the data to find patterns pales in comparison to an algorithmic approach.

Methods / Solutions

Beginning this project required establishing a base of knowledge about the dataset, submissions, and accuracy. The evaluative measure for a submission is AUC, or Area Under ROC curve, where a value can be between 0 and 1. A higher AUC means a better performance in prediction. The project language chosen was Python inside of Google Colab, using 'scikit-learn' for most of the regression on top of numpy and pandas. The first measure to establish a base of knowledge was submitting a valid 'dummy' submission of random predictions to understand what this prediction would yield. I generated random binary guesses for all 23,843 rows in the training data and submitted this prediction for a score of roughly 0.5. This means that a random guess is correct half the time, which is to be expected.

From here, an *in-progress high-level workflow* developed for my project, starting with finishing preprocessing. This is the high-level method for the project:

1. Preprocess data, gain baseline insight
2. Create / Train a new model
3. Evaluate the model
4. Find methods to improve
5. Repeat all steps

Preprocessing the data required loading the data into separate dataframes, and understanding general info like min, max, and mean using the '.describe()' method (*Figure 3*). The next step in preprocessing was data cleansing, which involved identifying missing values, outliers, and incorrect values. Blank values and '?' values were searched using '.info()', 'is.null()', and type counts. Incorrectly entered values were searched for in the categorical and numerical columns separately. Outliers were searched for using IQR. After preprocessing it was clear that there were no blank values (besides '?'), incorrectly entered values, or unreasonable outliers. Possible outliers like 'zero capital loss/gain' and '99 hours per week' were chosen not to be removed. Question mark values were replaced with averaged guesses using SimpleImputer. One extra imputation of 'holland' was made because this entry did not appear in the training dataset.

After gaining a baseline insight and preprocessing, it was time to attempt an initial model. Logistic regression was the first approach. The categorical columns in both datasets were converted to numerical values with pd.get_dummies so the model could use them. The 'scikit-learn' package was helpful for fitting the model to the training data and generating predictions using '.predict_proba', then rounding every prediction to either 0 or 1. Without rounding the predictions to 0 or 1, the first submission scored 0.57. After rounding my predictions, the next submission scored 0.61.

The next step was *finding methods to improve the model* before the next iteration. At the time it seemed reasonable to continue with logistic regression while implementing hyperparameter tuning (WandB, 2023). The training data was fit to a GridSearchCV object, and unfortunately the next predictions yielded the same AUC score as the untuned guess. My next method of improvement was scaling all the variables between 0 and 1, effectively normalizing the data before creating a model using a 'MinMaxScaler'. This also yielded no effect, so it was time to move on to a new approach.

On the next major iteration of the high-level strategy, a neural network was implemented. A binary classification task with thousands of data points seemed suited for a neural network to handle. First a 'StandardScaler' was used on the dataset to normalize the data. Second a model was used from Tensorflow titled 'Sequential'. A five layer neural network was used (*Figure 1*) with 1 Input layer, 1 dropout layer, 3 dense layers, with the final layer being a single output. The first dense layer was 256 neurons, and the second was 128 neurons. The dropouts were added to prevent overfitting. The activation function used was ReLU because of its commonality. The final activation function was sigmoid for the final output. The loss was calculated using 'binary_crossentropy', which is a logarithmic loss function for binary labels. The optimizer was titled "adam", which is an adaptive moment estimation algorithm using stochastic gradient descent (Ajagekar, 2021) . This first iteration of a neural network ran for 20 epochs. In the first epoch, the training data accuracy was 0.83. On the 20th epoch the accuracy was 0.87, which suggests that the accuracy did not improve substantially from the first iteration. The AUC score (accuracy for testing data) from Kaggle was 0.75, which is a significant improvement over the last model! However, the accuracy on testing data was far less accurate than on training data. This could suggest possible overfitting.

To combat potential *overfitting*, I decided to mess with the dropout parameter values. Increasing dropout from 0.25 to 0.5 yielded lower prediction accuracy. Decreasing dropout from 0.25 to 0.1 yielded a higher AUC of 0.76, from the original 0.75. This suggested that the model is underfit and the dropout layers could be removed. After removing the dropout layer a marginally higher score was achieved of 0.766.

To combat potential *underfitting,* the next attempt made was increasing the breadth and depth of the network (doubling the layers and doubling the 'Dense' value). This yielded a worse result, so attempts were made to lower both the density and depth separately. Neither yielded a higher performance. The next step was to test different hyperparameters of epochs, batch size, and learning rate. Higher epochs yielded lower AUC which suggests overfitting. Lower epochs yielded a lower as well, which means 20 iterations are around the sweet spot. A higher batch size yielded lower AUC, so the batch size of 35 was kept.

Results / Discussion

A summary of my results (*Figure 2*) shows the improvement over time from random guessing to logistic regression with non-binary predictions, to logistic regression with binary predictions, and ultimately to neural networks. From this summary it is clear the biggest jump came from switching the model, not just from correcting the model using hyperparameter tuning, averaging, etc. Ultimately neural networks yielded the most accuracy out of the three methods attempted. The 5 layer neural network with dropout was a significant improvement over the logistic regression. Lowering the dropout helped the model not generalize too much. With the architecture of this network, a certain accuracy / variance in the training data of around 0.9 percent accuracy seemed to be the peak before overfitting occurred in testing. Therefore, the 4 layer neural network with no dropout was the best performing network overall. This network overfit less than the 5 layer network with dropout.

In the final repository, I ended up removing the hyperparameter tuning code for the logistic regression model (GridSearchCV object) because it did not improve the model. Any code which did not significantly improve the model was removed, such as the MinMaxScaler method which tried to normalize the data for logistic regression. This helps the code consistency and conciseness.

If I had more time, I'd continue to explore the parameters which can improve my neural network, and perhaps try other methodologies like random forests. I would also find ways to make my model explainable, to understand which combinations of factors increase accuracy the most. I would also future explore how this model could be used for real world benefit.

References

Ajagekar, A. (n.d.). Adam. Adam - Cornell University Computational Optimization Open Textbook - Optimization Wiki. https://optimization.cbe.cornell.edu/index.php?title=Adam

Rmitson. (2023, November 30). Intro to MLOps: Hyperparameter Tuning. Weights &amp; Biases. https://wandb.ai/site/articles/intro-to-mlops-hyperparameter-tuning

Tables and Figures

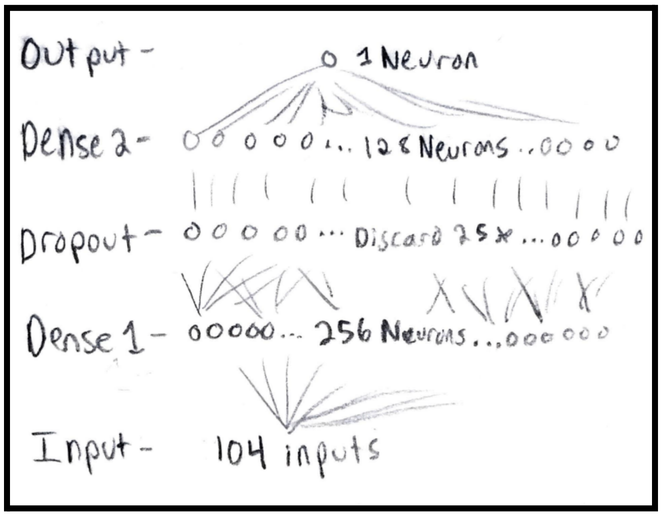


Figure 1

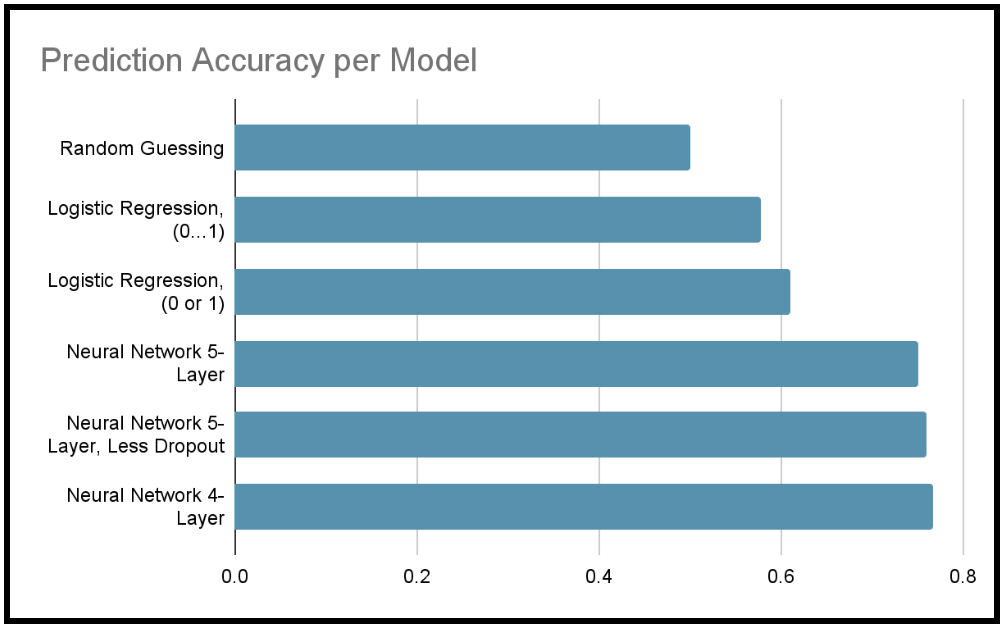
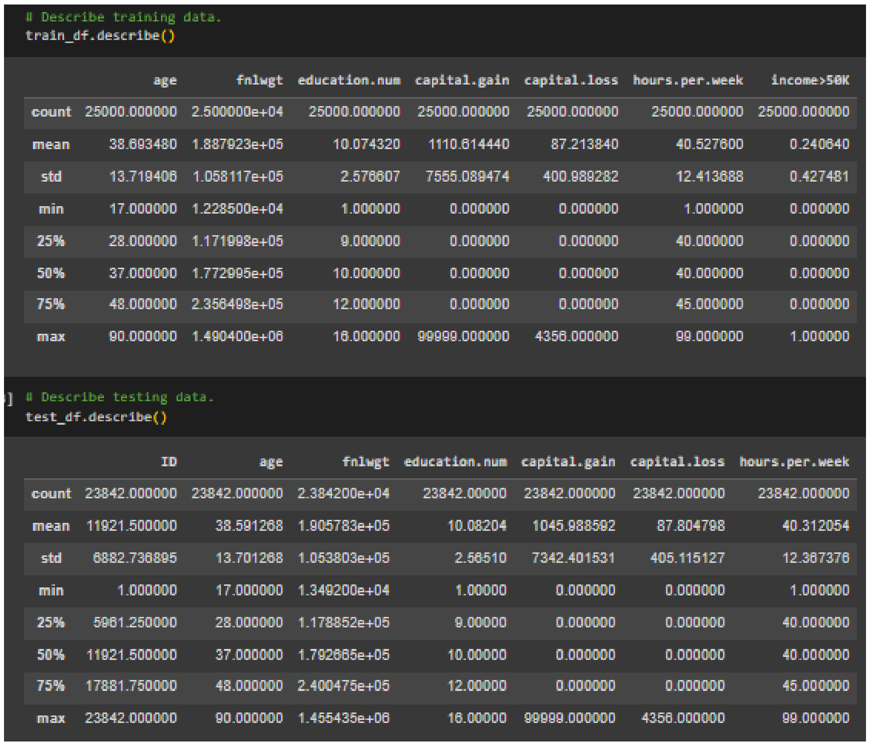


Figure 2



```
# Describe training data.
train_df.describe()
```

```
                age        fnlwgt  education.num  capital.gain  capital.loss  hours.per.week     income>50K
count  25000.000000  2.500000e+04   25000.000000  25000.000000  25000.000000    25000.000000   25000.000000
mean      38.693480  1.887923e+05      10.074320   1110.614440     87.213840       40.527600       0.240640
std       13.719406  1.058117e+05       2.576607   7555.089474    400.989282       12.413688       0.427481
min       17.000000  1.228500e+04       1.000000      0.000000      0.000000        1.000000       0.000000
25%       28.000000  1.171998e+05       9.000000      0.000000      0.000000       40.000000       0.000000
50%       37.000000  1.772995e+05      10.000000      0.000000      0.000000       40.000000       0.000000
75%       48.000000  2.358498e+05      12.000000      0.000000      0.000000       45.000000       0.000000
max       90.000000  1.490400e+06      16.000000  99999.000000   4356.000000       99.000000       1.000000
```

```
# Describe testing data.
test_df.describe()
```

```
                 ID           age        fnlwgt  education.num  capital.gain  capital.loss  hours.per.week
count  23842.000000  23842.000000  2.384200e+04    23842.00000  23842.000000  23842.000000    23842.000000
mean   11921.500000     38.591268  1.905783e+05       10.08204   1045.988592     87.804798       40.312054
std     6882.736895     13.701268  1.053803e+05        2.56510   7342.401531    405.115127       12.367376
min        1.000000     17.000000  1.349200e+04        1.00000      0.000000      0.000000        1.000000
25%     5961.250000     28.000000  1.178852e+05        9.00000      0.000000      0.000000       40.000000
50%    11921.500000     37.000000  1.792665e+05       10.00000      0.000000      0.000000       40.000000
75%    17881.750000     48.000000  2.400475e+05       12.00000      0.000000      0.000000       45.000000
max    23842.000000     90.000000  1.455435e+06       16.00000  99999.000000   4356.000000       99.000000
```

Figure 3