CS 5350/6350: Machine Learning Fall 2021

Alex Stewart Final Report
u1196342
https://github.com/AlexStew14/MLKaggle

Due date: 11:59pm, 19 Dec, 2021

1 Approach

1.1 Kaggle Background

Kaggle is a site used by Data Scientists to hone their craft and study new data analysis techniques in a similar way that a competitive programmer might use a site like HackerRank. Each kaggle competition has a fairly similar structure: a problem statement, provided training data, a submission scheme and a leader-board. For this competition the problem statement is predicting whether a given person will make more than 50k in income given some relevant information about the person. The data provided on each person consists of 14 features with categorical, continuous and binary data. Overall this competition is very similar to other competitions on kaggle and as I have done a few I already know which packages I will be utlizing.

1.2 Utilized Packages

For this competition I will be using python 3.9 with a few key packages for Data Science. First and foremost is Pandas which is used to read and process data with a minimal amount of code. Next is Matplotlib and Seaborn for data visualization which is a key part of any data analysis task as the goal of data analysis is gathering insights into data. Finally my model is a neural network built utilizing pytorch which is a library that facilitates deep learning with modular building blocks.

1.3 Data Preprocessing

For the midterm report, I was imputing missing values and utilizing label/binary encoding for categorical variables. For my final solution I instead took a different approach to missing values and categorical variables, feature embedding. Feature embedding is using a neural network to reconstruct categorical features with fewer dimensions than feature originally has. For instance, the native country field in our dataset has roughly 40 different categories which we would like to reduce to 20 dimensions. This is done by feeding the categorical

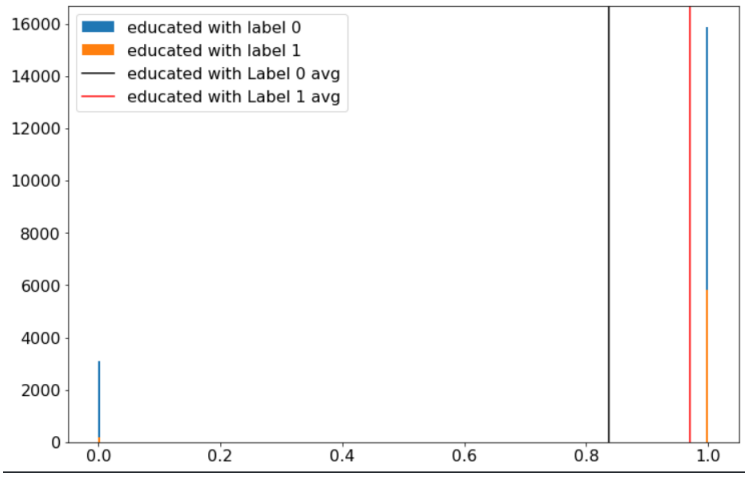

Figure 1: Labels for Educated and Non-educated people (num.education > 8)

data into a neural network with a hidden dimension of 20 that learns to reconstruct the input. Finally once the model is trained, the hidden layer can be used as a distilled version of the categorical data. To simplify this process, pytorch offers an Embedding layer which was used for all of the categorical features in the dataset. Missing fields were left in the dataset with the idea that the model might be able to utilize them after they are embedded. Aside from embedding categorical features, another key preprocessing step is downscaling the fnlwgt feature by subtracting the mean and dividing by the standard deviation. By reducing the scale of the fnlwgt feature, the stability of training was greatly improved which is likely because large input values means that slight changes in the corresponding weights drastically changes the layer output.

1.4 Feature Engineering

While feature engineering isn't as important for neural networks as is it is for other models due to the high level of expressivity, neural networks also benefit from having a higher number of useful features. For my model I added several new features which were mainly useful decision boundaries based on the feature distributions. The most useful decision boundaries I found are being a full time employee (hours.per.week ≥ 40) and being educated (num.education > 8) as can be seen in figures 1 and 2. Both of these figures show that being uneducated and part-time is highly correlated with making less than 50k. While adding these features didn't improve the trained model performance, I found that they helped the model converge more quickly.

1.5 Data Loading

A key insight from the dataset is the severe class imbalance in the label field, with around 75% of the samples being 0. With a dataset this imbalanced, often models will experience mode collapse where they fall into the local minima of only predicting the most common label as it will provide decent accuracy without much optimization. To circumvent this I

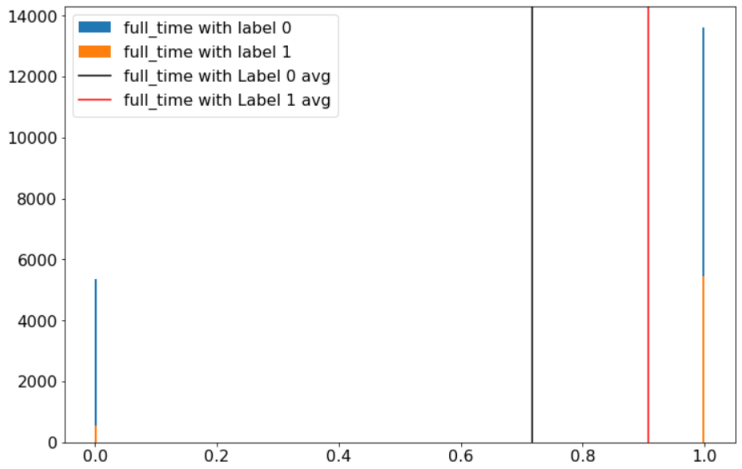

Figure 2: Labels for Full-time and Part-time employees (hours.per.week ≥ 40)

utilized pytorch's Dataset API to make a custom Dataset which assigned a weight to each sample that is the inverse of its frequency. Along with pytorch's WeightedRandomSampler and DataLoader pipeline, this ensured that each training batch had an even split of 0 and 1 label values which improved training stability and accuracy.

1.6 Model Architecture

As can be seen in figure 3, my model consists of three main blocks: input, internal and output. Before the input block, categorical features are fed into embedding layers and dropout is applied to help the model generalize as some categories for features like native country have very few sample points. Input and internal are identical aside from the input dimension to the linear layer while the output block has no batch normalization and a sigmoid activation layer. By making the model into three different blocks, the depth and width values can be easily varied with width determining the dimension of the linear layers and depth determining the number of times the internal block is repeated. For activation layers LeakyReLU is used to help prevent nodes from becoming deactivated which can happen with standard ReLU as ReLU has no slope for negative domain values. Sigmoid is used as the output activation to project the model's predictions to be between (0,1) which corresponds with the dataset labels.

1.7 Loss and Optimizer

For training the model I found that different optimizers and loss functions performed similarly with Binary Cross Entropy loss performing slightly better. This makes intuitive sense as this is a binary categorization problem and the range of the model is (0,1), whereas loss functions like L1 loss and Mean Squared Error loss also performed well but are typically used for regression problems. For optimizers I found that AdamW converged the quickest, but other optimizers like Adamax, Adam and RMSProp all resulted in very similar final results. To measure the model's performance aside from loss I used sklearn's area under

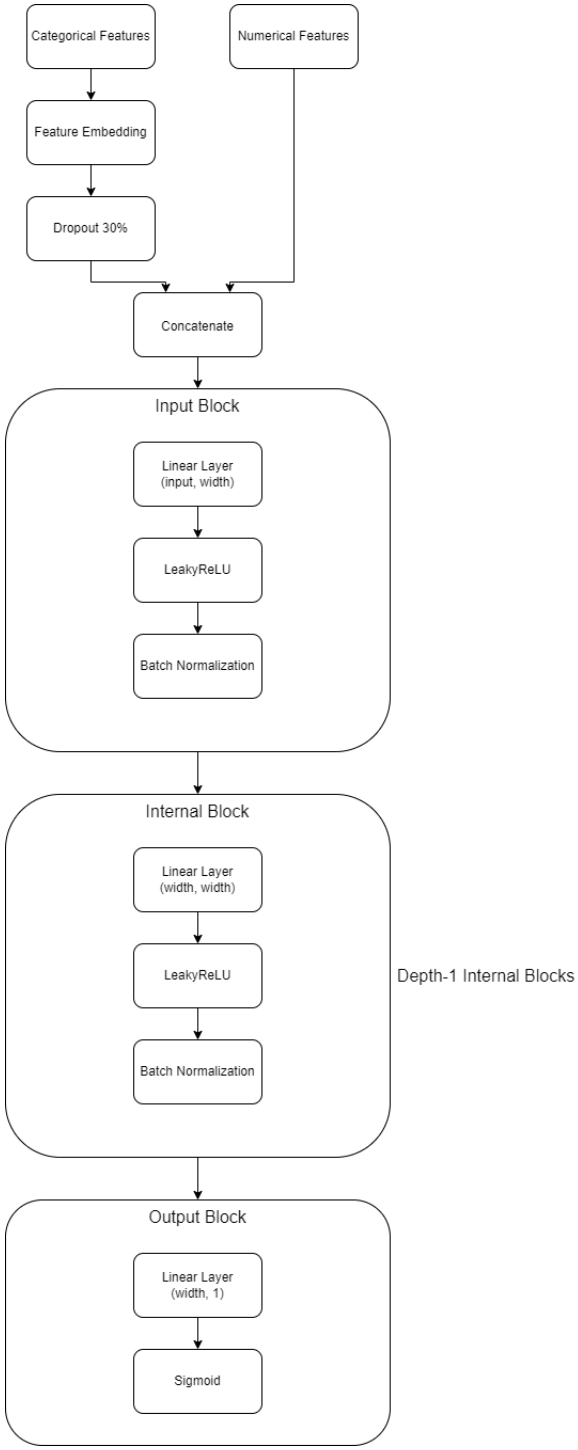

Figure 3: Model Architecture

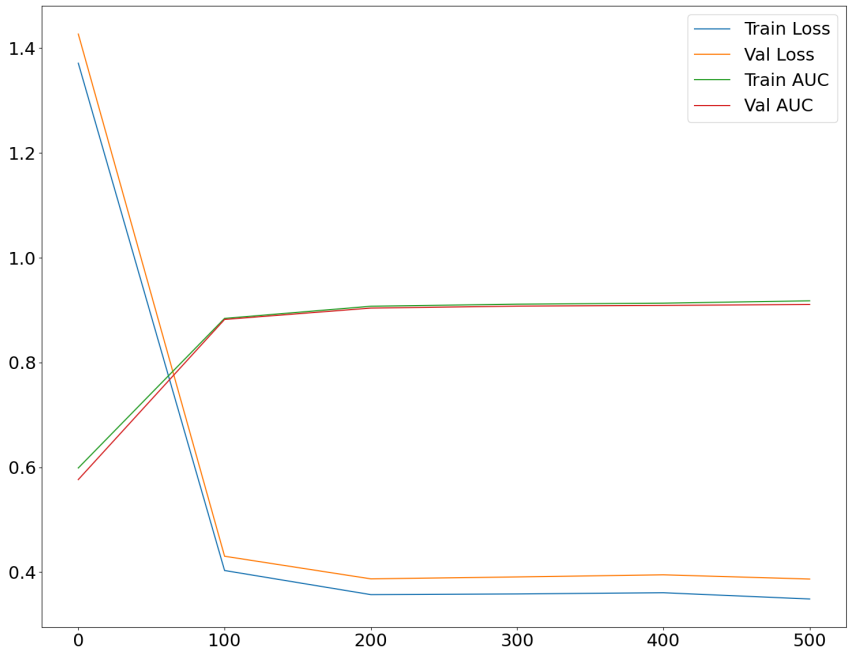

Figure 4: Cross Validation and Train Loss/AUC (Width 128, Depth 6)

receiver operating characterstic curve method as it is the same accuracy measure used for the kaggle leaderboard. Switching from error to this metric resulted in much closer leaderboard performance compared to my cross validation scores. In addition, by not submitting rounded values (ie 0 or 1) I found that my leaderboard score went up significantly. This is a result of using area under curve instead of error as the model will give the expected value for each sample which results in on average a better score.

1.8 Training and Cross Validation

To help determine the optimal parameters for my model I perform some very rudimentary cross validation. Currently I set aside 20 percent of the training data for cross validating and then train the model on the remaining 80 percent. As can be seen in figure 4 by the AUC lines, the model generalizes extremely well to the validation data with the train AUC being only slightly higher than the validation AUC. As can be seen in figure 5, varying the depth and width results in extremely similar results for each combination. This suggests that the function the model learns from mapping the input to an output prediction is simple enough that a shallow network is able to learn it. My leaderboard submissions are based off of a model with width 128 and depth 6, but it's clear that in this case any combination tested would perform comparably.

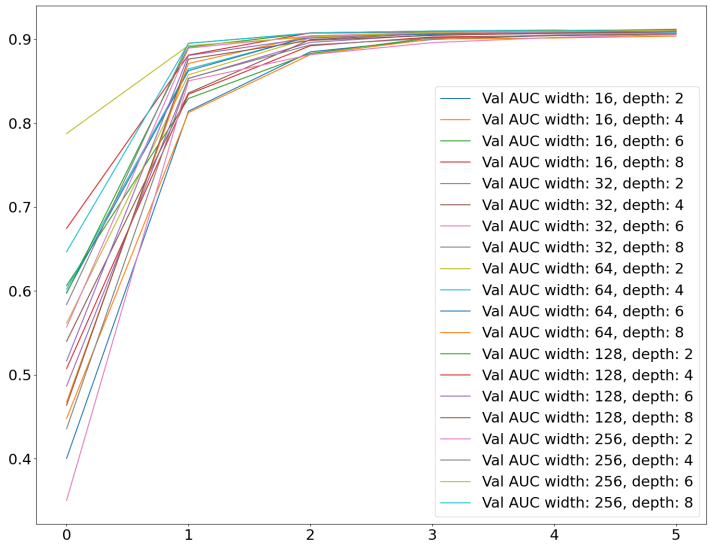

Figure 5: Validation AUC for different architectures

1.9 Going Forward

Although I'm happy with my final performance (.91406 leaderboard AUC), with more time I would have liked to experiment with model ensembling. Since gradient boosting techniques like XGBoost or CatBoost seem to perform well on this problem, with more time I would try gradient boosting with feature embedding and utilizing both my neural network and a boosting model together to produce the final submissions. In addition, while I experimented with a few different activation layers and adding dropout to my internal block, with more time I would perform more rigorous testing on different architectural changes to my model.