Hamed Hekmat, Nina Boord, Michael Brockman, Iris Fu

Startup Success Predictor

Introduction

Will AI replace venture capitalists? Or perhaps will AI soon become a necessary tool to give venture capitalists an edge in the market? In our final project, we implemented neural networks to predict whether or not a startup will become successful based on data points about the startup's location, funding history, and various other factors. This problem is important because although venture capitalists often do their qualitative due diligence, they primarily use their intuition to make investment decisions. Computers, however, excel at sifting through extensive historical data and may provide insight into implicit, counterintuitive trends. Such trends, although they should not determine an investment, may facilitate venture capitalists in making more informed decisions. For example, our model found that startups that raised their first funding earlier—rather than later—on were more likely to fail. This is one of the many possible findings that some venture capitalists might find counterintuitive but helpful in their decision-making.

Overall, we found our neural network approach to provide more accurate predictions than our logistic regression baseline. For our model, we conducted the following steps: preprocessing, splitting (into training, validation, and test datasets), logistic regression (as a baseline), neural network preprocessing, hyperparameter tuning, neural network training, and neural network evaluation. In this paper, we will go through each step and its requirements.

Literature Review

A working paper we hoped to build upon is *Using Deep Learning to Find the Next Unicorn: A Practical Synthesis* (Cao, Lele, et al). This paper gives a literature review and synthesis of DL methods of startup evaluation. This was helpful in inspiring considerations. Namely, determining startup success criteria including material success vs. investment compatibility concerning investor goals, and addressing data imbalance with far more failures than successful ones. Additionally, there is a discussion about what aspects of a startup should be included in a prediction. The paper details prior implementations to this problem, including using ML (such as decision trees and logistic regression) and DL with artificial neural networks and hidden layers. Our work will be complementary to this work by directly comparing different models, namely logistic/linear regression and a small neural network. With this paper's framework, our small NN—which will produce potentially complex relationships in the data—should perform much better than the simple logistic regression.

In addition, we were inspired by *Using Machine Learning to Demystify Startups Funding, Post-Money Valuation, and Success* (Ang, Yu Qian, et al). This research group utilized Dirichlet allocation and a neural network to predict startup success (hitting benchmarks such as IPO or acquisition) via variables such as geography, description, sector, number of investors in their region, etc. Our work will sit adjacent to their ML/neural network research, and we would also like to consider additional variables such as race and gender.

Dataset

Many datasets had high-quality data about companies that still exist, which would have been difficult to classify as a "success" or "failure" as we do not know their outcome. Companies at their meteoric peak have failed and companies on the brink of collapse return to still be relevant today. Additionally, some datasets had many features but did not have all of them on

each company, resulting in many null data points. This would have made it difficult to make substantial connections between companies and success. To have the most accurate and measurable results, it was important to choose a dataset with a conclusive classification for success and few missing data. Thus, we chose Startup Success Prediction by Manish KC, which had a focused scope on whether a company was acquired or closed.

The dataset contains 923 companies and 49 features, including their industry, founding state, funding rounds (including when they chose to raise and whether it was an angel investor or venture capital), number of milestones and when they met them, number of founders, and number of industry connections the company had. We used if the company was acquired or closed as the label. There were 597 companies labeled as "acquired" and 326 companies labeled as "closed."

Before we began training our model, we centered every feature to mean 0, decided how to handle certain null values, and excluded unhelpful features. It was important to center each feature as we wanted our variance to be around some point to have the most high-quality conclusions when comparing different companies. The only null values that existed in the dataset were related to the "milestone" feature, where if a company never completed a milestone, then their age for completing that milestone was empty. In this case, we decided to set that value to be arbitrarily large as it took them an infinite amount of time to complete it. Additionally, we excluded highly specific and non-quantitative features such as company name, latitude, longitude, and city. Finally, we excluded when the company closed (if the company closed) because that would be an obvious indicator that the company had closed, and likely be the strongest predictor. The pre-processing and reading of data is done in our *read* function. Then, the *splitDataset* function divides the data into training, validation, and testing sets (using an

80-20 split to partition the data into a training set and a temporary set, then splitting the temporary set into equal parts to form the validation and testing sets). The function returns the split datasets which are ready for model training and evaluation.

Recognizing potential inequity is important in data selection. For instance, white men have historically had an advantage in raising money in the first place. Thus, a variable such as funding profile might skew the results such that companies founded by white male founders are predicted to have better outcomes. This could reinforce existing inequalities and contribute to an unfair cycle where models are predicting that minority founders will be unsuccessful due to their inability to raise funds, thus contributing to their inability to raise funds.

Baseline

We implemented logistic regression for our baseline using scikit-learn (see function *logReg(X_train, y_train, X_val, y_val, X_test, y_test)*). The model's coefficients and intercept are printed out, which provides insight into the relative importance of each feature. The model is then evaluated on the validation and test sets, with the accuracy and other classification metrics printed out. We also investigated the final weights after training (more in Results and Analysis).

Main Approach

We used a PyTorch neural network with cross-entropy loss with an Adam optimizer.

First, we used our training and validation set with 10 epochs to optimize the hyperparameters, namely the learning rate, hidden layer depth, hidden layer width, and batch size. We chose the hyperparameters to test by taking an educated guess and finding local optima. For example, if we chose to test learning rates [.1, .01, .001], and the optimal learning rate was .001, we would test .0001. If that did not improve accuracy, we know we can stay at .00l. With this approach, it is essential to only test one variable at a time. Ultimately we tested learning

rates of [0.1, 0.01, 0.001, 0.0001, .00001], batch sizes of [16, 32], and hidden layer configurations = [[5], [64], [5, 4], [64, 32], [5, 4, 3], [64, 32, 16]]. While this was a time-intensive process, we believe this led to more successful results.

After these hyperparameters were set, we used 10 epochs to train and then ran it on the test set, measuring items on our evaluation metric.

Evaluation Metric

Our evaluation metric consisted of test accuracy, precision, recall, and F1 score. Test accuracy measured the proportion of true results (both true positives and true negatives) among the total number of cases examined, or otherwise characterized as the overall success. Precision measures the proportion of positive identifications that were correct, this would be important for an investor because sinking significant capital into a company means that the cost of a false positive is high. Recall measures the proportion of actual positives that were correctly identified, which would be useful if the cost of a false negative was high, which may not be the case here. F1 score is a common harmonic mean of precision and recall, calculated as: $2 \times \frac{recall \times precision}{recall + precision}$.

Results and Analysis

For our baseline logistic regression model, we saw a test accuracy of 62.37%, precision of 0.62 , recall of 0.6, and F1 score of 0.6. In analyzing the weights after training, we noted some interesting trends. For example, it was better to have a larger gap between the founding date and the first fundraising round, implying that it was better to wait for funding, rather than to fundraise right away. We also observed (unsurprisingly) that when a startup has more funding they are more likely to get acquired. The best state to found a successful startup was California, and the worst was Texas. The best industry was software, and the worst was e-commerce. Having VC rather than angel investor funding was also a better predictor of startup success.

As for the results of our main approach, our designed hyperparameter tuning process proved to be imperfect as it yielded highly variable results. Considering all configurations of our various hyperparameters, each run would generate and test 60 possible neural network architectures, the performance of which ranged from around 20% accuracy to over 80% accuracy on the validation set. Unsurprisingly, the best-performing configuration of hyperparameters outperformed our baseline almost every single time. However, due to the aforementioned issue of variability, some runs (albeit very rarely) would output final "best fit" hyperparameters that were evaluated to perform slightly worse than the baseline. In all of our iterations and runs, the hyperparameter combination we observed to yield the absolute best performance was a learning rate of 0.1, batch size of 32, and hidden layer configuration of [5, 4]. The final trained neural network with these hyperparameters had an accuracy of 86.21%, with a precision of 0.74, recall of 0.86, and F1 score of 0.8. While not perfect, this optimal neural network architecture (at least among the subset of potential hyperparameters we tested) demonstrated significantly better performance over the baseline.

For other qualitative observations of the effects of various hyperparameters on performance, we noted an interesting dynamic between the depth/width and the learning rate. First, more complex architectures (3 hidden layers vs 1 or 2, or many more neurons per hidden layer) performed worse as a whole, perhaps a reflection of the fact that this is a relatively simple classification problem. Unnecessarily robust architectures likely overfit the training data, leading to worse performance on unseen data points. However, on runs where the configurations with 3 hidden layers performed relatively better, we noticed that the best-performing configurations were those that paired 3 hidden layers with a much smaller learning rate (e.g. 0.0001), perhaps implying the slower learning rate dampened the overfitting effect. This intuition also aligns with

the fact that our best-performing configuration had a relatively simpler architecture (perhaps a "goldilocks" moment at 2 hidden layers without very many neurons in each) paired with the highest learning rate we used (0.1). As a final note, the effect of batch size on performance seemed to be negligible, as runs with different batch sizes yielded very similar results. Perhaps this relates to the relationship between the batch sizes we used and the size of our dataset.

Error Analysis

Our initial system involved manually modifying the configuration of hyperparameters used by the neural network constructor, leading to a lot of trial and error done by hand. Thus, after some tampering and experimenting to get a general sense of what kinds of values we wanted to test for our hyperparameters, we decided to add the hyperparameter tuning function to avoid rerunning an individual neural network every time. This was probably the most important experiment we ran and the realization we had that influenced the direction of our final design as it significantly reduced the manual labor involved in changing the inputs, running an individual model, and recording performance metrics to compare with past runs.

However, this decision also introduced the largest issue with our final project. As mentioned in the previous section, we observed immense variability in the output of our hyperparameter tuning function from one run to the next, exposing its sensitivity to the overall stochasticity of the neural network initialization and training process. For example, our neural network class initializes the architecture with random weights and biases, already affecting the performance of identical configurations in various runs. Additionally, considering we have a relatively small dataset, the randomness involved in data batching during the training process further implies two runs with the same hyperparameter configurations will likely perform

differently. Pairing these considerations with the potential existence of noisy data points or unnecessary features in the dataset further exacerbates the variability of results between runs.

Future Work

Alongside resolving the aforementioned issue of our program's sensitivity to the stochasticity of the training process, we could expand this project by using different datasets and confirming some of our findings on various data sources. We also determined that we could make the data less noisy by removing features irrelevant to our main objective, such as "founding date." However, we would have to think critically about these features and remove them one at a time to see if the model becomes more accurate. For instance, there could very well be an interplay between the founding date and the type of startup, as the market tends to go through cycles as money is pumped into a certain industry during a particular time (web3, AI, etc.), so more trial and error is needed to make any decisions about removing features. Furthermore, PyTorch utilizes a specific optimizer to adjust the weights of the network during training. The one we used was the Adam function from the torch.optim library, is only one of many possible optimization algorithms. Other optimizers we could test out include SGD, which is a classic optimizer that updates parameters in the direction of the negative gradient, and RMSprop (Root Mean Square Propagation), which adapts the learning rates based on the moving average of squared gradients. As discussed prior, the primary challenge we would like to address looking forward is finding a way for our results to be stable rather than stochastic.

In conclusion, while we have not quite replaced venture capitalists with AI (yet), we have determined that AI is a powerful tool that can likely give users an edge in making investment decisions. Given our results, we think that the future of these types of predictions will likely be in the realm of neural nets and not classical statistical methods like logistic regression. All this

said, qualitative observations, past experiences, and intuitions are all still important tools that VCs use when making investment decisions. AI insights may inform these decisions, but care must be used due to the ethical implications of reinforcing existing inequities in tech entrepreneurship.

Code - https://github.com/ninaboord/startup-predictor

References

Ang, Yu Qian, et al. "Using machine learning to demystify startups funding, Post-Money valuation, and success." *SSRN Electronic Journal*, 2020, https://doi.org/10.2139/ssrn.3681682.

Cao, Lele, et al. "Using Deep Learning to Find the next Unicorn: A Practical Synthesis." *arXiv.Org*, 18 Oct. 2022, arxiv.org/abs/2210.14195.