Predicting Startup Success: A Machine Learning Approach

Thomas Styron
Brown University Undergraduate
github.com/tstyron/startup-predictor

Introduction
This machine learning project aims to predict startup success, defined as acquisition by a larger company, using supervised learning. Given the high failure rate (90%) of startups and the rise of crowdfunding, this project seeks to offer computational risk assessment tools for investors.

The data, sourced from CrunchBase and uploaded to Kaggle, includes 54,294 datapoints and 39 columns, encompassing categorical, date/time, and numerical data. The 'status' column, indicating whether a startup is acquired, operating, or closed, serves as the target variable.

EDA
Baseline:
There are 54,294 datapoints and 39 columns. Of the 39 columns, there are ten categorical columns, six date/time columns, and 23 numerical columns.

Categorical overview:
Of the ten categorical columns, there are three that function as unique identifiers for the companies. One column is the target variable (‘status’). The remaining six categorical columns are related to the companies’ industries/markets (2) and their location (4).

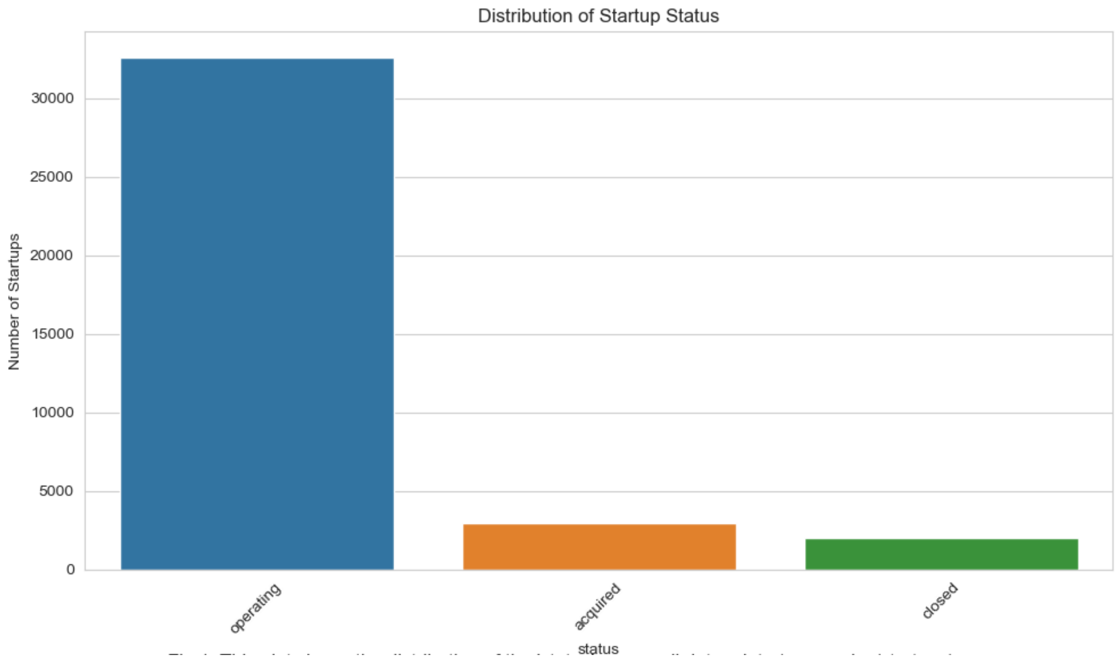
Date/Time Columns:
The six date and time columns pertain to when the company was founded (year, quarter, month, day) (4), and when their first (1) and last funding occurred (1).

Numerical Columns:
Of the 23 numerical columns, 21 relate to the dollar amount of funding the company received in 21 funding rounds. One column is the number of funding rounds the company participated in, and one column is the company’s total amount of funding.

EDA Continued:
Missing Values:
To proceed with the data exploration, I determined the percentage of null values in each of the remaining 48,123 columns. All missing values are in categorical columns. 38.6% of ‘state_code’ are null, 21.9% of all four of the columns pertaining to when the company was founded are null. Eight other columns have missing values and are all under 13% null.

Figures:
In my analysis, I first examined the distribution of companies by 'status,' revealing a significant data imbalance: 86.9% operating, 7.7% acquired, and 5.4% closed. This was followed by an investigation into how total funding varied across these statuses, showing that acquired startups typically received the most funding, followed by operating and then closed startups. Furthermore, while acquired startups generally participated in more funding rounds, there was no

notable difference in the number of funding rounds between operating and closed companies. Finally, my regional analysis highlighted that a majority of the startups in the dataset are U.S.-based, particularly in the Bay Area, New York, and Boston.



[illegible]

Figure 2: Distribution of [illegible]

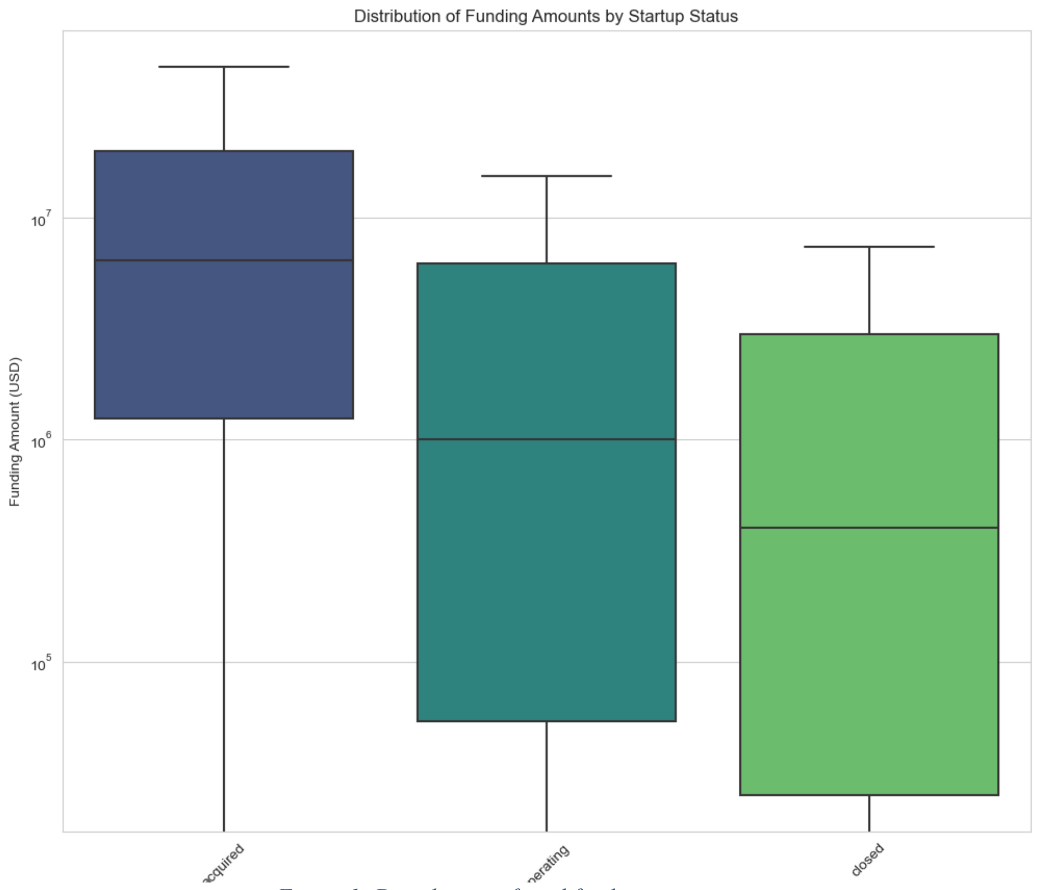


Figure 1: Distribution of total funding across status.

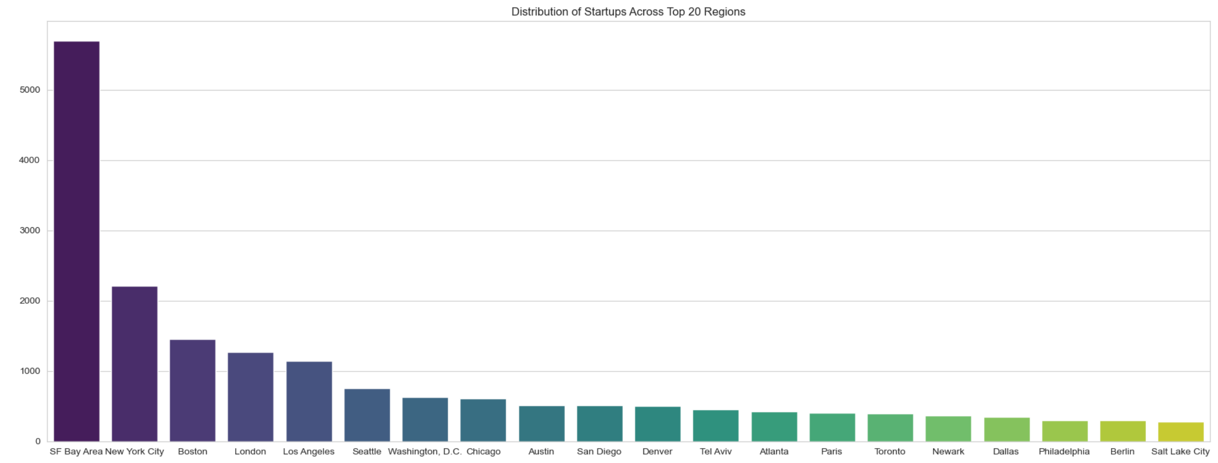


Figure 3: Distribution of startups across regions.

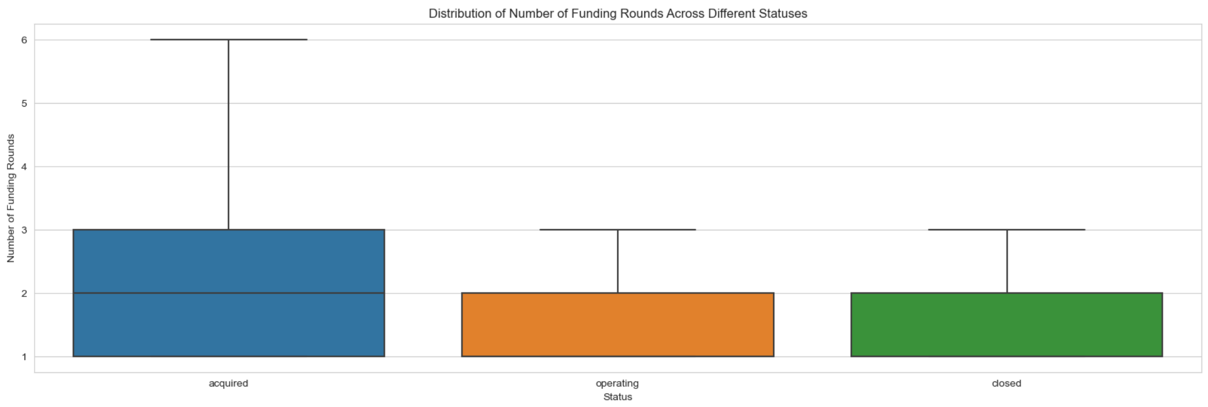


Figure 5: Distribution of funding rounds across status

Feature Engineering

To enhance the predictive accuracy of models, new features were generated from existing data columns. Four numerical columns were created: 'diff_funding_time' and 'diff_funding_months' (measuring the time between each company's first and last funding round in days and months, respectively), 'diff_first_funding_months' (the time from a company's founding to its first funding round), and 'industry_group' (groupings of 'market' categories). These differential columns aim to replace static time identifiers, focusing on relative rather than absolute timings, to better generalize for future data. In training the XGBoost model, two additional columns were derived from another dataset: 'num_investors' (the count of unique investors) and 'avg_investment' (the average investment per investor). However, due to data mismatches causing around 40% missing values, these columns were challenging to integrate into other models without advanced imputation methods.

Methods

Splitting

When splitting the data for training and testing, 60% of the data was used for training, 20% for validation, and 20% for testing. To address the large data imbalance, a stratified split was used – ensuring that each split of the data has roughly the same number of each target class. Stratified split was used in tandem with K-fold cross validation to further enhance the model's ability to generalize and to ensure a robust evaluation. The use of K-fold cross-validation, in this case, allowed for the assessment of model performance across multiple subsets of the data, reducing the impact of any specific partitioning on the model's performance metrics.

Preprocessing

Numerical Columns:

To preprocess the data before training, numerical columns were normalized to have a mean of zero and standard deviation of one using scikit-learn's StandardScaler(). This ensures a reduced impact of outliers on the performance of the model and improve convergence times.

Categorical Columns:

A one-hot encoder was used on numerical columns to ensure that categorical columns could be trained on.

Evaluation Metric:

In assessing model performance and tuning hyperparameters via scikit-learn's GridSearchCV, two methodologies were considered: an f1 micro score and a custom scoring metric averaging f1 score on only the 'acquired' and 'closed' target classes. The latter, aligning with stakeholder priorities in evaluating investment risks in operating companies, was preferred. This approach emphasizes predictive accuracy for companies likely to be acquired or fail, as opposed to merely operating. However, accurately identifying operating startups remains crucial; a model indiscriminately predicting 'acquired' or 'closed' status would misleadingly score higher than trained models. Therefore, a balanced approach, potentially employing an f1 macro score, is suggested for future model training, to be elaborated in the Outlook section.

Models

The models we trained are Logistic Regression, Random Forest, SVC, K Nearest Neighbors, and XGBoost. The models and the respective hyperparameters that were tuned are in the table below. The class_weight parameter was used for all models where applicable, with each target classes' weight proportional to its representation in the data.

Table 1

Model	Hyperparameters	Description
Logistic Regression	C: [0.01, 0.1, 1, 10, 100]	Regularization strength; smaller values specify stronger regularization.
Logistic Regression	Solver: [liblinear, lbfgs]	Algorithm used for optimization
Random Forest	Max Depth: [1, 3, 5, 10, 20, 30, 50, 80, 100, 200, None]	Maximum depth of the trees; controls overfitting.

SVC	C: [0.1, 1, 10, 100]	Regularization parameter; controls the trade-off between smooth decision boundary and classifying training points correctly.
SVC	Gamma: [scale, auto]	Kernel coefficient for 'rbf', 'poly', and 'sigmoid'; influences the decision boundary.
K-Nearest Neighbors	N Neighbors: [3, 5, 7, 9, 12, 15, 20, 40, 80]	Number of neighbors to use; affects the decision boundary.
K-Nearest Neighbors	Weights: [Uniform, Distance]	Weight function used in prediction; 'uniform' weights all points equally, 'distance' gives closer points higher weight.
XGBoost	Max Depth: [3, 5, 7]	Maximum depth of a tree; increasing this value will make the model more complex and likely to overfit.
XGBoost	Learning Rate: [0.001, 0.01, 0.1, 0.15]	Step size shrinkage used to prevent overfitting; a lower rate requires more boosting rounds.

Uncertainties in Model Performance:
In the analysis of uncertainties in the evaluation metric due to splitting, a Logistic Regression model across three different random states was utilized to examine the variability in performance during the cross-validation pipeline. The results revealed mean F1 scores of approximately 0.163, 0.160, and 0.163 with corresponding standard deviations of 0.048, 0.049, and 0.047 across four folds for the three random states, respectively. These figures indicate a relatively consistent average performance of the model across different data splits, as shown by the close mean F1 scores. However, the standard deviations highlight a non-negligible level of variability in the model's performance across the folds within each split. This variability, albeit moderate, suggests that the model's performance is somewhat sensitive to the specific manner in which the data is partitioned.

Regarding the variability in test scores associated with non-deterministic models like XGBoost and Random Forest, the analysis revealed no notable differences in score variance across various random states when compared to linear models. This suggests a comparable level of stability in performance between these non-deterministic and linear methodologies.

Results

Scores/Metrics:
In assessing the performance of various models against the established baseline score of approximately 0.109 (custom f1 score), derived from a naive model predicting company statuses with equal probability at random, several models exhibit varying degrees of predictive accuracy. The Logistic Regression model achieves a mean test score of approximately 0.203, surpassing the baseline but falling short of more advanced models. The Random Forest Classifier records a mean test score of about 0.247. Similarly, the Support Vector Classifier (SVC) presents a mean test score of around 0.222. The K-Neighbors Classifier, however, underperforms with a mean test score of approximately 0.102, slightly below the baseline. Among these, the XGBoost (xgb) classifier emerges as the most predictive model, with a mean test score of roughly 0.268, significantly outperforming the baseline and other models in predictive accuracy.

Inspection of XGBoost's confusion matrix reveals further insight into the model's performance on the test set. The model achieved 48% accuracy on the *acquired* target class, 42.5% accuracy on the *closed* target class, and 73.8% accuracy on the *operating* target class, revealing a strong tradeoff between accuracy on minority classes and the majority class, despite our custom f1 metric's lack of consideration for the performance on the majority class.

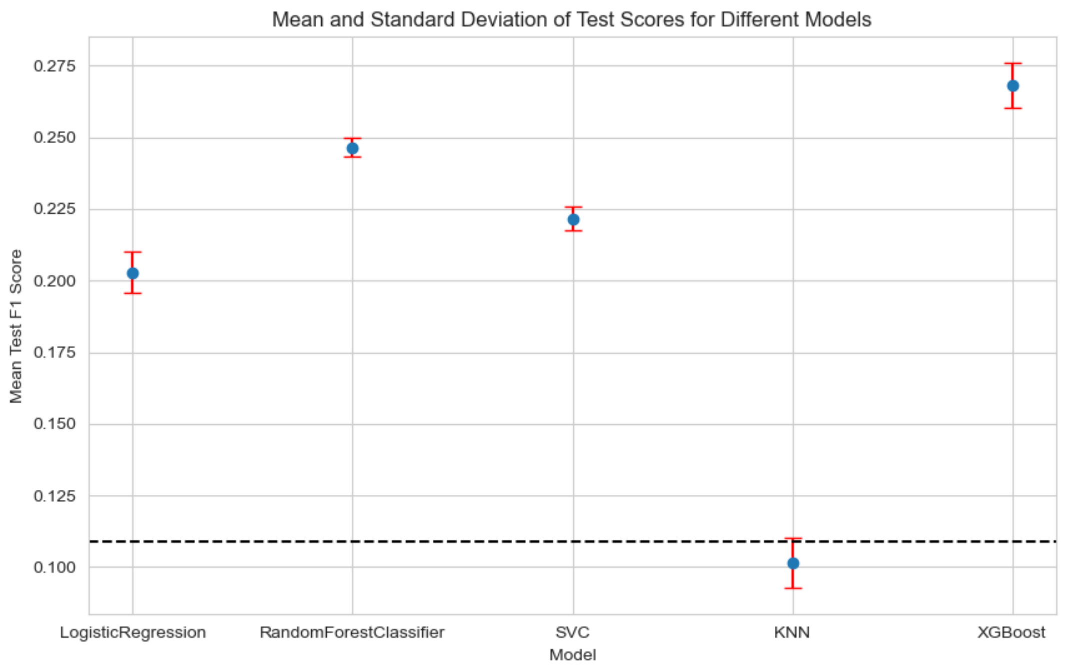


Figure 6: Error bars depicting mean and standard dev. of model custom f1 test scores. Dashed line represents baseline score.

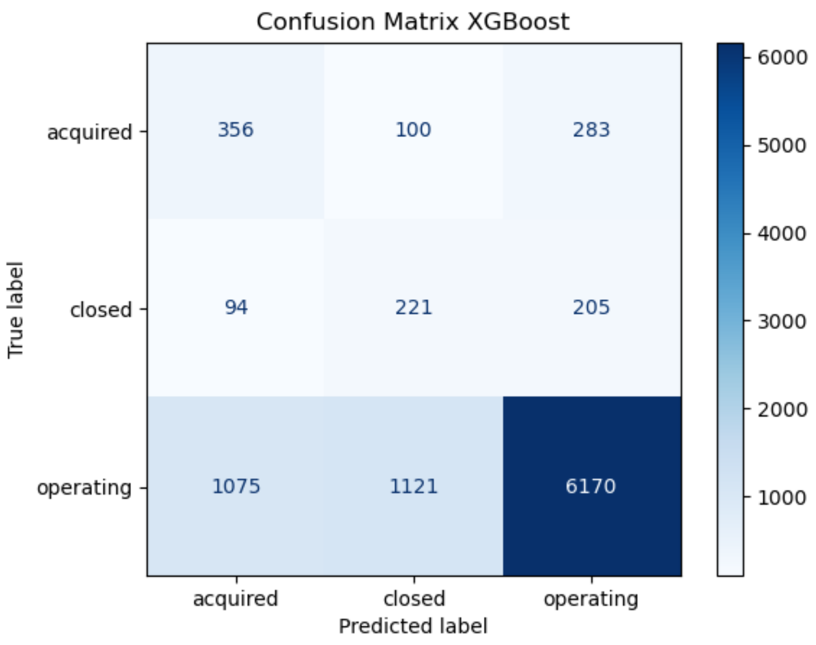


Figure 7: Confusion matrix for XGBoost on the test set.

XGBoost Interpretability and Feature Importances:
In determining feature importance for the XGBoost model, three methods were used to calculate **global feature importance**: gain, cover, and SHAP values. From these figures, several points of comparison emerge. The SHAP values indicate a balanced distribution of feature importances with 'avg_investment' and 'funding_total_usd' being most significant, which suggests a more nuanced impact of each feature on model predictions. In contrast, the Gain metric shows a steep initial drop, with 'num_investors' and 'country_code_USA' dominating, indicating these features contribute most to the increase in model performance. The Cover metric reveals a similar pattern to Gain, with 'cat_industry_Group_Science and Engineering' and 'cat_resources' leading, suggesting these features are responsible for covering a larger number of instances in the data. Notably, there is little overlap in the top features across the three methods, highlighting the distinct interpretative angles each method provides, with SHAP focusing on average impact, Gain on performance increase, and Cover on instance coverage.

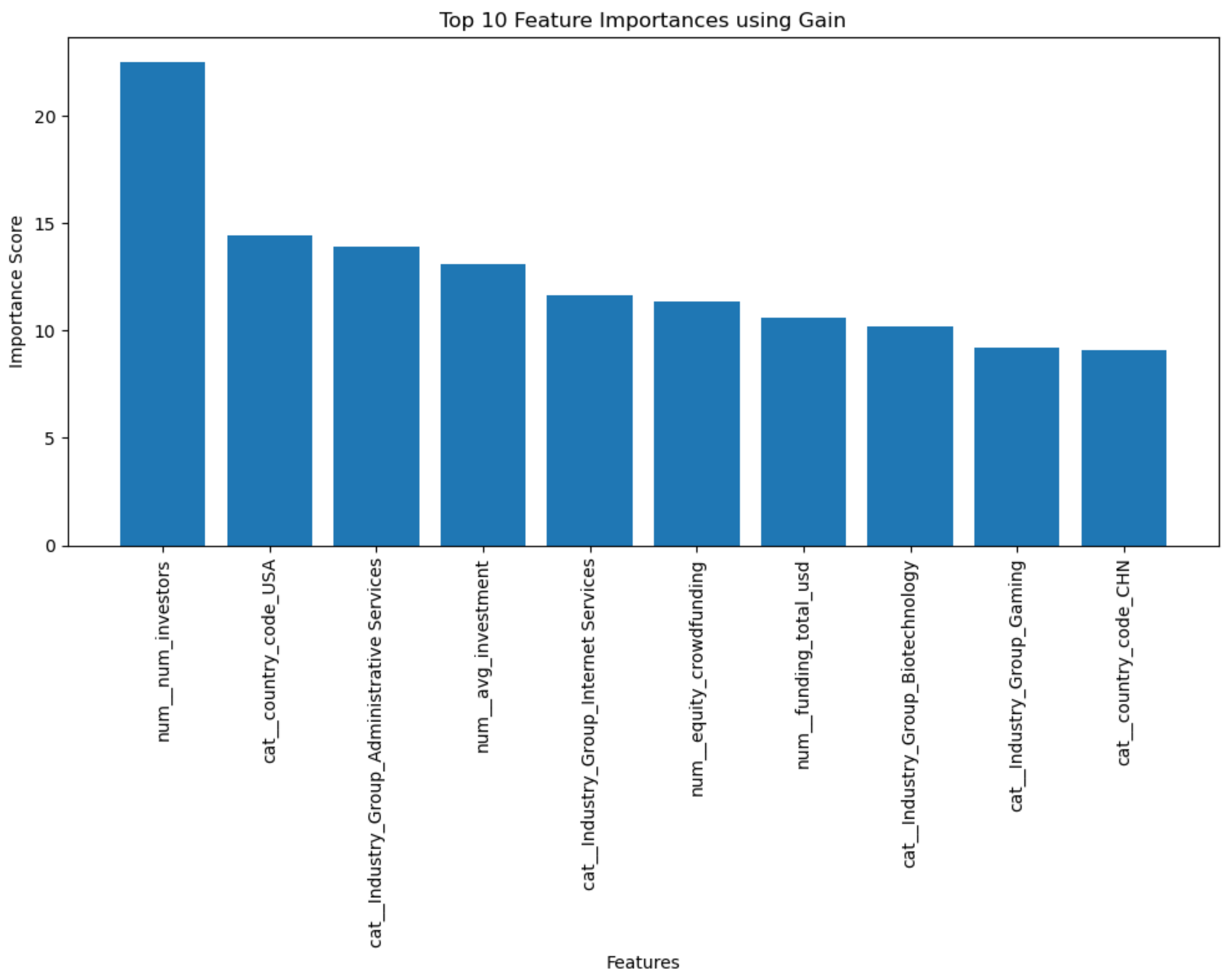


Figure 8: Top 10 global feature importances for XGBoost model calculated using Gain.

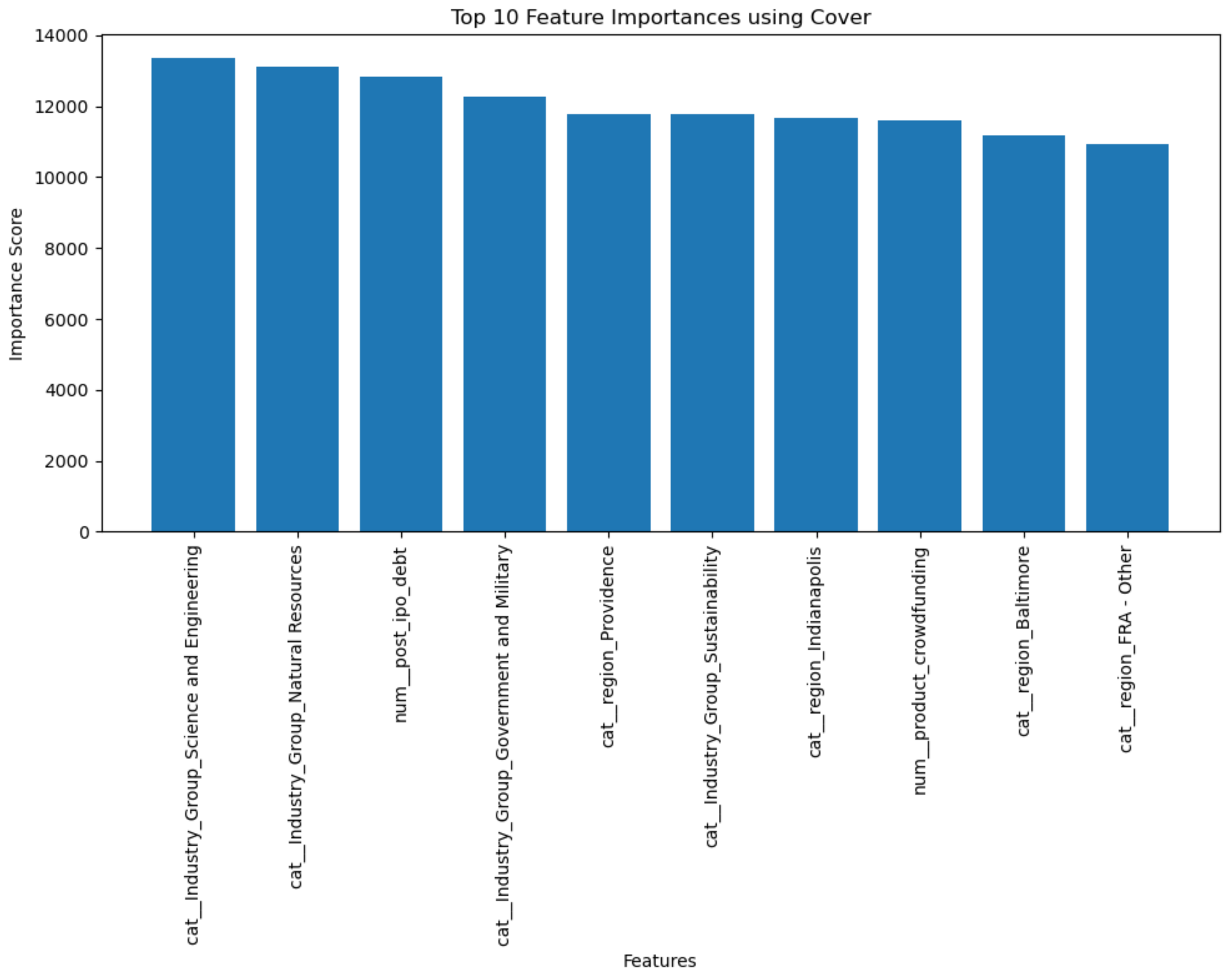


Figure 9: Top 10 global feature importances for XGBoost model calculated using Cover.

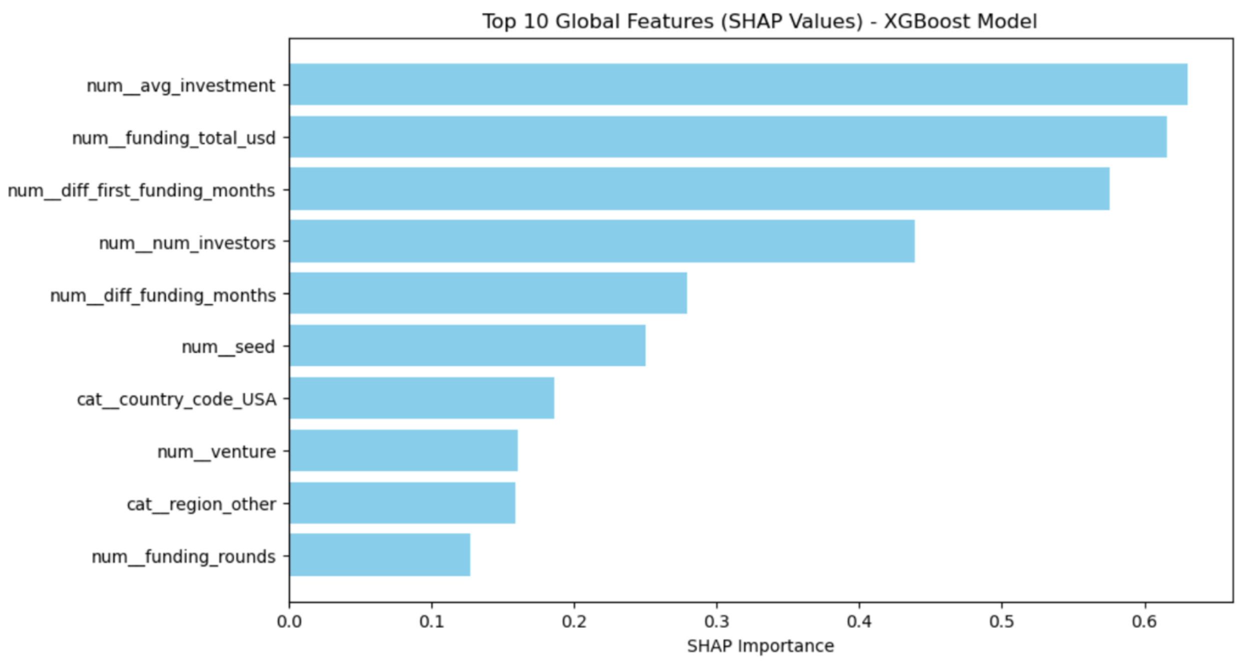


Figure 10: Top 10 global feature importances for XGBoost model calculated using SHAP values.

In examining **local feature importance** for three distinct companies (acquired, closed, operating), SHAP values were separately calculated to generate SHAP force plots, accurately predicted by the model. These plots, using red values to show features positively influencing predictions and blue for negative influence, revealed a higher predictive confidence for the operating class, aligning with its greater representation in the training data.

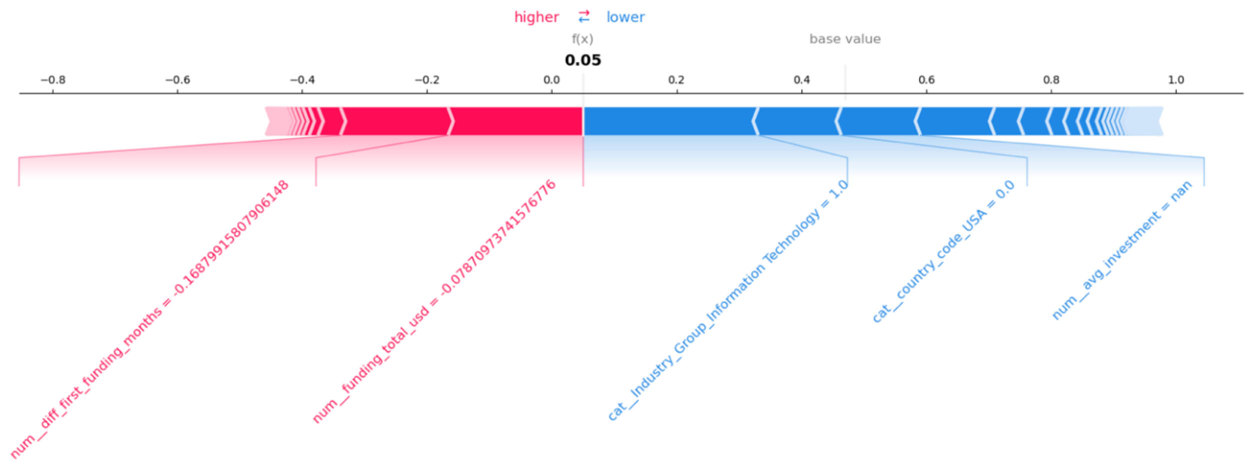


Figure 11: Local feature importance and SHAP values for 'acquired' target class.

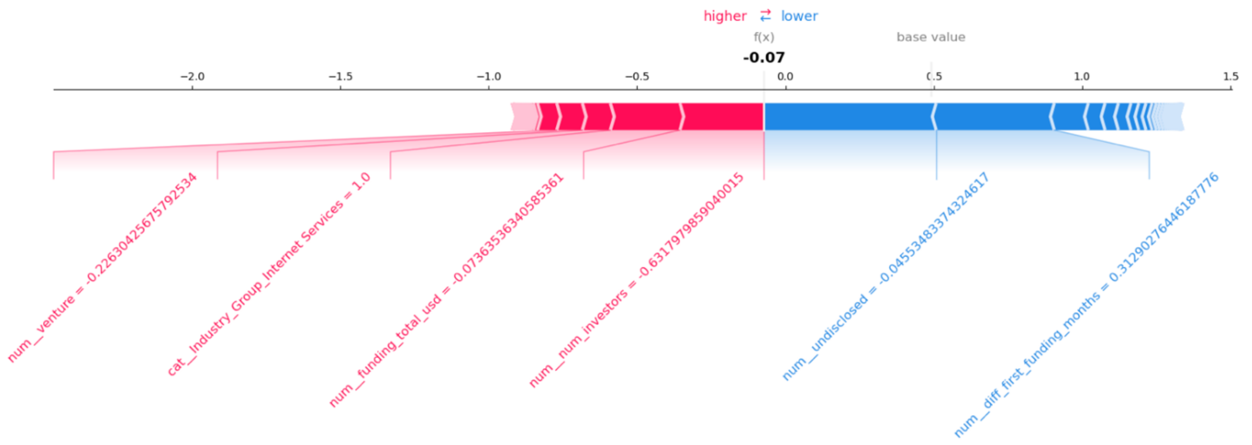


Figure 12: Local feature importance and SHAP values for 'closed' target class.

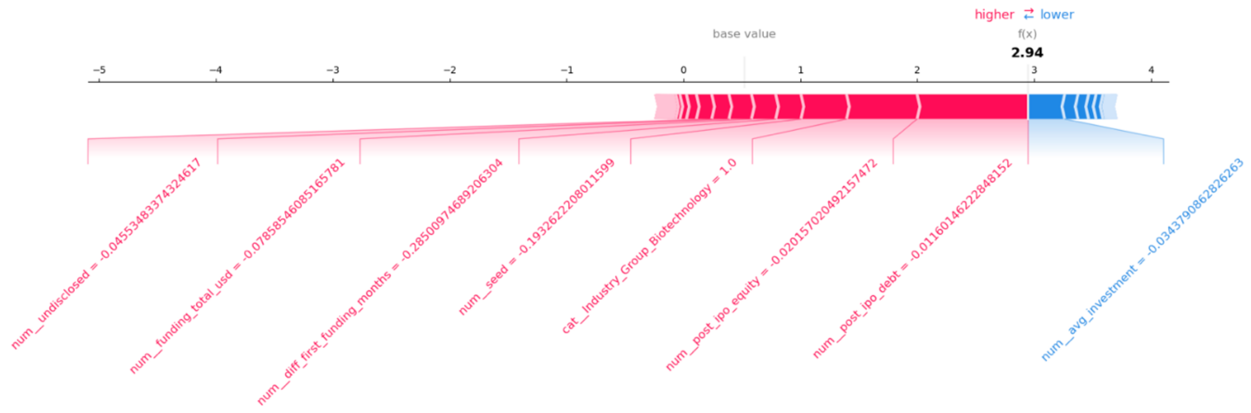


Figure 13: Local feature importance and SHAP values for 'operating' target class.

The analysis revealed that the engineered features, avg_investment and num_investors, were frequently influential in both local and global contexts. Surprisingly, seed and venture funding rounds, being initial funding stages commonly pursued by startups, also emerged as significant in SHAP feature importance analyses. This prominence contrasts with the expectation that later funding rounds, given that participation is more scarce and thereby more indicative of continued success, would exert greater influence on the model's predictions.

Outlook

Scoring

Using f1 macro score to train the model, as this would weight all target classes equally and give weight to the *operating* target class. This may lead to a better tradeoff between correctly predicting *acquired* and *closed* class, while also giving weight to the *operating* class.

Missing values

Different approaches to handle missing values (besides the use of XGBoost which can train on null values) such as multivariate imputation or training reduced feature models would allow for all models to be trained and evaluated using the 'avg_investment' and 'num_investors' columns. (More in Feature Engineering section).

New Data
Seeing as we cannot rely on the date a company was founded or funded alone (as described in Feature Engineering ection), an improvement would be to include macroeconomic data associated with the times that a company was founded or raised money. Similarly, new data on the founders of these companies (education level, previous startups, stake in company, etc.) could be leveraged to improve predictive power.

References

Initial Kaggle dataset uploaded by user arindam235:
https://www.kaggle.com/datasets/arindam235/startup-investments-crunchbase

Uploaded by Justinas Cirtautas, investments.csv from Kaggle was used in conjunction with initial dataset for feature engineering leveraged by XGBoost model:
https://www.kaggle.com/datasets/justinas/startup-investments

Credit to Prianka Ball, whose code was used to generate the Industry_Group column and for correcting existing column names and datatypes:
https://www.kaggle.com/code/pball01/predicting-startup-success