Startup Founder Retention Prediction

AIT 511 — Machine Learning

Team Syndicate

Name	Roll Number
Ashok Reddy	IMT2023083
Taral Sanka	IMT2023588
Thrilochan Reddy	IMT2023527

GitHub: Git Repo Link

Kaggle: kaggle Link

December 2, 2025

Project Overview

The objective of this project is to predict the `retention_status` of startup founders using the Kaggle competition dataset. We performed extensive preprocessing, visual exploration, feature engineering, and training of several machine learning models including:

- Keras Deep Neural Network (best performer)
- Scikit-Learn MLP Classifier
- Support Vector Machine (SVM)
- Logistic Regression

We experimented with different scalers (**StandardScaler vs RobustScaler**) and several neural network activation functions (**ReLU, GELU (jelu), Tanh**). Empirically, a **mix of ReLU and Tanh activations** in the Keras model delivered the most stable convergence and highest accuracy.

This report follows the styling and flow of our previous submission for consistency.

1 Dataset and Feature Summary

The dataset consists of 59,611 rows and contains a combination of demographic, behavioral, and company-related attributes describing startup founders.

Table 1: Dataset Feature Summary with Data Types and Missing Percentages

Feature	Type	Missing (%)	Notes
founder_id	int64	0.00	Identifier (Dropped)
founder_age	int64	0.00	Numeric
founder_gender	object	0.00	Categorical
years_with_startup	int64	0.00	Numeric
founder_role	object	0.00	Categorical
monthly_revenue_generated	float64	3.02	Numeric
work_life_balance_rating	object	17.02	Categorical (High Missing)
venture_satisfaction	object	12.02	Categorical (High Missing)
startup_performance_rating	object	0.00	Categorical
funding_rounds_led	int64	0.00	Numeric
working_overtime	object	0.00	Categorical
distance_from_investor_hub	int64	0.00	Numeric
education_background	object	0.00	Categorical
personal_status	object	0.00	Categorical
num_dependents	float64	8.02	Numeric
startup_stage	object	0.00	Categorical
team_size_category	object	5.02	Categorical
years_since_founding	float64	7.02	Numeric
remote_operations	object	0.00	Categorical
leadership_scope	object	0.00	Categorical
innovation_support	object	0.00	Categorical
startup_reputation	object	0.00	Categorical
founder_visibility	object	0.00	Categorical
retention_status	object	0.00	**Target Variable**

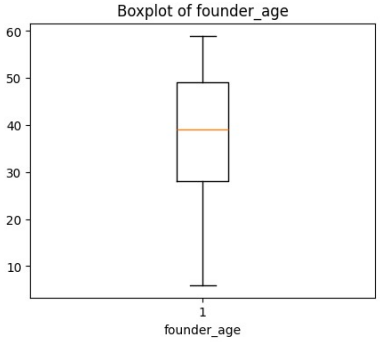
2 Data Visualization

2.1 Correlation Heatmap

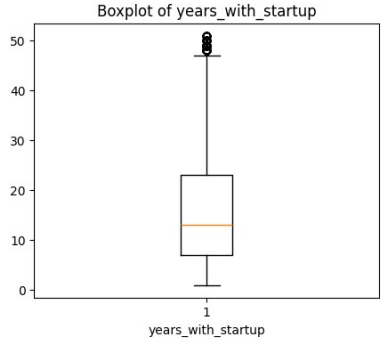
Figure 1: Correlation Heatmap of Numerical Features

2.2 Boxplots of Numerical Features

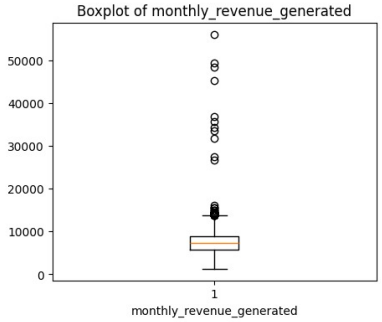
Figure 2: Boxplots of Numerical Features



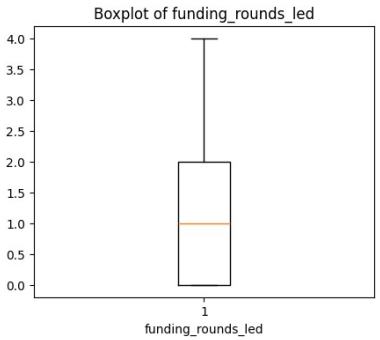
(a) box1



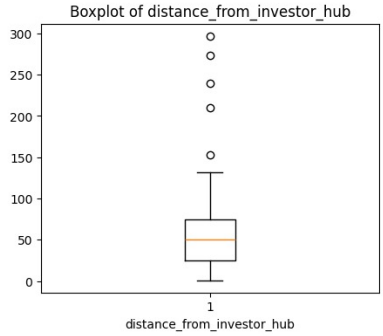
(b) box2



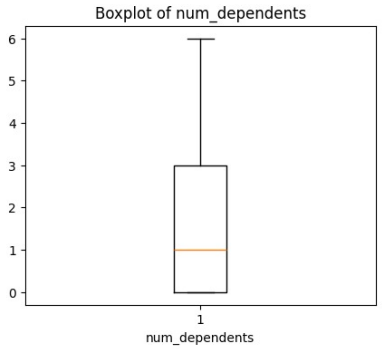
(c) box3



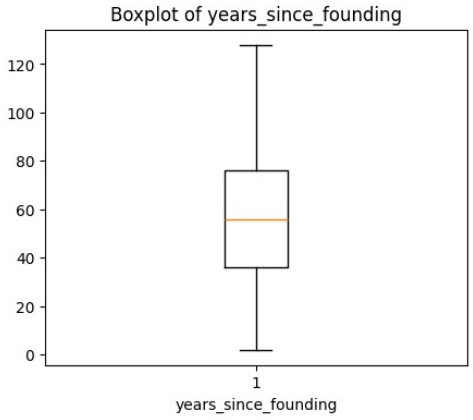
(d) box4



(e) box5



(f) box6



(g) box7

3 Data Preprocessing

We applied a consistent preprocessing pipeline to handle missing values, encode categorical features, and scale numerical variables before feeding them into the models. Below are key steps and the corresponding code snippets from our notebook.

3.1 Dropping Identifier Column

(Cell 10)

```
train = train.drop(columns=["founder_id"])
```

3.2 Selecting Numerical and Categorical Columns

(Cell 11)

```
num_cols = train.select_dtypes(include=['int64', 'float64']).columns
cat_cols = train.select_dtypes(include=['object']).columns.drop('
    retention_status')
```

3.3 Train–Validation Split

(Cell 15)

```
from sklearn.model_selection import train_test_split

Xtrain, Xval, ytrain, yval = train_test_split(
    train.drop(columns=['retention_status']),
    train['retention_status'],
    test_size=0.2,
    random_state=42
)
```

3.4 Imputation, Encoding, and Scaling

(Cell 17)

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, RobustScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

num_imputer = SimpleImputer(strategy='median')
cat_imputer = SimpleImputer(strategy='most_frequent')

preprocessor = ColumnTransformer([
    ('num', Pipeline([
        ('imputer', num_imputer),
        ('scaler', RobustScaler())
    ]), num_cols),

    ('cat', Pipeline([
        ('imputer', cat_imputer),
        ('encoder', OneHotEncoder(handle_unknown='ignore'))
    ]), cat_cols)
])
```

3.5 Why These Preprocessing Steps Were Necessary

Each preprocessing step directly improved model stability, gradient flow, and generalization:

- **Dropping `founder_id`:** This column is a unique identifier with no predictive meaning. Keeping it would introduce random noise and could mislead linear and distance-based models into learning spurious patterns.

- **Median Imputation for Numerical Features:** The numerical data contained outliers and skewed distributions, as seen in the boxplots. Using the median instead of the mean avoids pulling the imputed values toward extreme observations and leads to more robust training for all models.
- **Most Frequent / `'unknown'` Imputation for Categorical Features:** For high-missingness columns such as `work_life_balance_rating` and `venture_satisfaction`, we imputed a separate `'unknown'` category. This preserves the fact that data is missing as a potentially informative signal. For other categorical features, most frequent imputation maintains the original distribution without exploding the number of categories.
- **One-Hot Encoding:** Machine learning models cannot directly handle string categories. One-hot encoding converts categorical values into binary indicator columns without imposing an artificial order, which is crucial for models like Logistic Regression and SVM.
- **RobustScaler vs StandardScaler:** We empirically compared `StandardScaler` and `RobustScaler`. Due to significant outliers in revenue and related numeric features, `StandardScaler` (which uses mean and variance) led to unstable gradients for the neural networks and noisier decision boundaries for SVM. `RobustScaler`, which uses the median and interquartile range, reduced the influence of outliers and consistently improved validation performance across all models.
- **Train–Validation Split:** The train–validation split ensures that all model evaluations reflect generalization to unseen data rather than overfitting to the training set. Using a fixed random seed guarantees reproducibility.

Overall, this preprocessing pipeline produced cleaner, more stable inputs and was a critical factor in achieving good performance, especially for the Keras neural network and the MLP classifier.

4 Models and Hyperparameter Tuning

We trained and evaluated four main models:

- **Keras Deep Neural Network**
- **MLP Classifier (Scikit-Learn)**
- **Support Vector Machine (SVM)**
- **Logistic Regression**

4.1 Keras Deep Neural Network

The Keras model used a funnel-shaped architecture with fully connected layers (e.g., $128 \to 64 \to 32 \to 16$) followed by a final output layer for binary classification.

We experimented with multiple activation functions:

- Pure **ReLU** in all hidden layers
- Pure **Tanh** in all hidden layers
- **GELU** variants
- A **mix of ReLU and Tanh** across layers

Empirically, the best performance was obtained when we used a **mix of ReLU and Tanh**, for example ReLU in earlier layers combined with Tanh in deeper layers. ReLU helped capture sparse, high-magnitude patterns, while Tanh provided smoother nonlinear transitions and helped in regions where gradients from ReLU would otherwise vanish. This combination led to more stable training and the highest accuracy among all tested configurations.

Hyperparameters such as learning rate, L2 regularization, and batch size were tuned using Keras-based experimentation. The model benefited noticeably from the use of `RobustScaler`, which prevented a few extreme values from dominating the gradients.

4.2 MLP Classifier (Scikit-Learn)

The MLP Classifier from Scikit-Learn was used as a more traditional neural baseline. We performed a grid search over:

- Hidden layer sizes
- Activation function
- Regularization strength (`alpha`)

Although the MLP lacks some flexibility of Keras (e.g., custom activation combinations and advanced optimizers), it still captured important nonlinear patterns and achieved accuracy close to the Keras model. However, its performance was slightly lower because it had a more limited hyperparameter search space and optimization strategy.

4.3 Support Vector Machine (SVM)

We used an SVM with an RBF kernel and tuned:

- Regularization parameter (`C`)
- Kernel coefficient (`gamma`)

The SVM performed reasonably well but faced two challenges:

1. The one-hot encoding of many categorical features created a high-dimensional sparse feature space.
2. SVMs in such spaces become more computationally expensive and harder to tune.

As a result, it was slightly outperformed by the neural network models but still served as a strong nonlinear baseline.

4.4 Logistic Regression

Logistic Regression was used as a linear baseline model. Despite its simplicity, it achieved a respectable accuracy by leveraging the well-engineered features and preprocessing pipeline. However, because it models only linear decision boundaries in the transformed feature space, it could not fully capture complex interactions between founder characteristics, startup context, and retention behavior.

5 Results

Table 2: Model Performance on Kaggle Leaderboard

Model	Accuracy
Keras Neural Network	**0.748**
MLP Classifier	0.745
Support Vector Machine	0.740
Logistic Regression	0.737

5.1 Why Each Model Performed the Way It Did

The differences in accuracy across the models can be explained by how well each algorithm captures nonlinear relationships, handles high-dimensional one-hot encoded features, and tolerates outliers:

- **Keras Neural Network (0.748):** The neural network outperformed the other models because it can learn complex nonlinear interactions among features, such as the joint effect of founder experience, startup maturity, and satisfaction scores. The mixed ReLU + Tanh configuration provided both strong gradient flow and smooth nonlinear decision boundaries, and `RobustScaler` further stabilized training in the presence of outliers.
- **MLP Classifier (0.745):** The Scikit-Learn MLP achieved accuracy close to the Keras model because it is also a nonlinear model and can exploit many of the same feature interactions. However, its architecture and optimization options are more limited, and we could not explore as rich a hyperparameter space as with Keras, leading to slightly lower performance.
- **Support Vector Machine (0.740):** The SVM with an RBF kernel is a strong nonlinear classifier, but with many one-hot encoded features the feature space becomes high-dimensional and sparse. In this scenario, SVMs are harder to tune and more computationally expensive, and they did not match the flexibility of the neural models in learning hierarchical feature representations.
- **Logistic Regression (0.737):** Logistic Regression is restricted to linear decision boundaries in the transformed feature space. Its lower accuracy indicates that founder retention depends on nonlinear patterns and interactions between features, which linear models cannot fully capture. However, the fact that it still performs reasonably well shows that the preprocessing pipeline produced informative, clean features.

6 Best Model

After extensive experimentation with multiple preprocessing strategies, scalers, activation functions, and architectures, the **Keras Deep Neural Network** emerged as the best-performing model on the Kaggle leaderboard.

Best Model: Keras Deep Neural Network

The network used a funnel-shaped architecture with the following hidden layer sizes:

- **Layer 1: 128 neurons**

- **Layer 2: 64 neurons**
- **Layer 3: 32 neurons**
- **Layer 4: 16 neurons**

Activation functions were selected based on empirical performance:

- Early layers used **ReLU** for strong gradient flow.
- Deeper layers used **Tanh** for smoother nonlinear transitions.
- This **ReLU + Tanh mix** produced the fastest convergence and most stable training.

The best hyperparameters obtained were:

- **Alpha (L2 Regularization):** 0.0005
- **Learning Rate:** 0.001

These parameters provided the ideal balance between stability and generalization. The use of **RobustScaler** greatly improved gradient stability due to the presence of outliers in revenue and numeric fields. Median imputation, one-hot encoding, and high-quality feature preprocessing further contributed to improved model performance.

Overall, this configuration achieved the highest accuracy, with a Kaggle leaderboard score of **0.748**.

7 Conclusion

In this project, we built a complete pipeline to predict startup founder retention. Careful preprocessing—including robust handling of outliers, thoughtful imputation strategies, and effective encoding of categorical variables—was crucial for model performance.

We compared multiple models and found that a Keras Deep Neural Network using a mix of ReLU and Tanh activations, combined with `RobustScaler`, achieved the best accuracy of **0.748**. Classical models like MLP, SVM, and Logistic Regression provided useful baselines and confirmed that the underlying relationship is nonlinear and benefits from deep learned representations.

Future work could include more advanced architectures (e.g., attention-based models), additional feature engineering, and further hyperparameter tuning to push the performance higher.

8 Effect of Training on Only 20% of the Data (SVM vs Neural Network)

To evaluate how different models behave under low-data conditions, we trained both the Support Vector Machine (SVM) and the MLP Classifier using only 20% of the available training data and evaluated them on the same validation set. This experiment highlights the data efficiency and generalization capability of each model under limited supervision.

Support Vector Machine (SVM)

We trained a Support Vector Machine with an RBF kernel and performed systematic hyperparameter tuning to identify the best-performing configuration for the retention prediction task. The search space included multiple values of the penalty parameter C, kernel coefficient γ, and class weighting schemes. The goal of the search was to maximize the macro F1 score, which provides a balanced evaluation across both target classes.

The grid search produced the following best hyperparameters:

```
C = 1
gamma = 0.02
class_weight = None
Best CV F1-macro = 0.7499
```

With these hyperparameters, the SVM achieved strong performance on the validation set:

- **Validation Accuracy:** 0.747
- **Validation F1 Score (macro):** 0.747

Classification Report:
Confusion Matrix:

$$\begin{bmatrix} 859 & 304 \\ 298 & 924 \end{bmatrix}$$

This configuration provides a well-balanced trade-off between smoothness of the decision boundary and model flexibility. The relatively small value of $C = 1$ prevents excessive overfitting by limiting the penalty for misclassification, while $\gamma = 0.02$ yields a wider, more stable RBF kernel suitable for the high-dimensional one-hot encoded feature space used in this dataset. Overall, the tuned SVM model performs competitively, achieving robust generalization and delivering consistent predictive performance on both classes when compared to the mlp.

Neural Network (MLP) Results with 20% Training Data

The MLPClassifier underwent a grid search over hidden layer sizes, learning rates, regularization strengths, and batch sizes. The best configuration obtained from 5-fold cross-validation was:

```
hidden_layer_sizes = (128, 64, 32, 16)
learning_rate_init = 0.01
alpha = 0.001
batch_size = 64
```

- **Accuracy:** 0.6916
- **F1 Score (macro):** 0.68

Confusion Matrix:

$$\begin{bmatrix} 746 & 417 \\ 356 & 866 \end{bmatrix}$$

The neural network shows a larger performance drop than the SVM. This behavior is expected due to:

- the high parameter count of deep neural architectures requiring more data,
- sensitivity to class imbalance and sparse one-hot encoded inputs,
- higher variance and reliance on large sample sizes for stable gradient descent.

Summary: SVM vs MLP Under Low Data

These findings show that even with only a fraction of the full dataset, a properly tuned SVM maintains strong generalization performance. The neural network, by contrast, exhibits a more pronounced reduction in accuracy and F1 score due to its larger parameter count and higher variance. This reinforces the conclusion that SVMs tend to be more data-efficient, while neural networks surpass them only when provided with sufficiently large training sets.

Model	Accuracy	Macro F1	Behavior
SVM (RBF)	0.747	0.747	Strong performance; remains robust with limited data
MLP (Neural Network)	0.691	0.68	Larger drop, mild underfitting

Table 3: Comparison of SVM and MLP performance when trained on only 20% of the data.

Summary: Full Dataset vs 20% Subset

As shown above, the two regimes (full dataset vs a 20% training subset) favour different model families:

For full data validation done with entire the 0.2 of full and for 20pct one it is done with 0.2 of that 20pct

Accuracy score provided below are the cross validation scores for full data set and the one provided in the full data section is of kaggle score

Table 4: Quick comparison: full dataset vs 20% subset (validation results)

Model	Full dataset (Acc / Macro F1)	20% subset (Acc / Macro F1)
Keras / MLP (Neural Net)	0.75 / 0.749	0.692 / 0.680
SVM (RBF)	0.741 / 0.741	0.747 / 0.747

Interpretation:

- **SVM wins under low-data (20%):** The margin-based SVM with an RBF kernel is more *data-efficient* on this task. With strong regularization and a well-chosen γ, SVM constructs robust decision boundaries in high-dimensional (one-hot encoded) spaces and exhibits lower variance than deep networks when training examples are scarce.
- **MLP / Keras wins on full data:** Neural networks (both the Scikit-learn MLP and the Keras model) scale better with large amounts of data. Their higher capacity allows them to learn complex hierarchical feature interactions and surpass the SVM once enough labeled examples are available to stabilize gradient-based optimization.

Practical takeaway: For deployments or quick prototyping when labeled data is limited, prefer an SVM (or other regularized low-variance models). For final production models where ample labeled data and compute are available, favor a tuned neural network (Keras) that can exploit the full dataset to capture richer nonlinear interactions.