CS634 Final Term Project Report

Yash Shah
CS634 – 101
Dr. Yasser Abduallah

Telecom Churn Prediction Data Using Random Forest, Conv 1D, and SVM
Nov 10, 2025

1. Introduction

The purpose of this project is to build, evaluate, and compare multiple machine learning models for binary classification. Specifically, the goal is to predict customer churn (whether a telecom customer will stay or leave), based on demographic, account, and service-related features. This helps businesses identify at risk customers and take proactive retention actions.

Binary classification is a supervised learning task where the model predicts one of two possible outcomes (e.g., Yes/No, Churn/Not Churn, Fraud/Not Fraud). It's important because many real-world decisions rely on such two-class predictions which helps enable targeted marketing, risk management, medical diagnosis, and customer retention strategies. Accurate binary classification helps optimize business decisions and resource allocation.

Dataset: Telco Customer Churn Dataset which contains customer demographics, service subscriptions, billing details, and churn status (Yes/No).
Algorithms Implemented:
Random Forest (mandatory): An ensemble learning method that builds multiple decision trees and averages their results for robust predictions.
Conv1D (Deep Learning Model): A 1D Convolutional Neural Network that captures local feature patterns and nonlinear relationships in tabular data.
Support Vector Machine (SVM): A classic machine learning algorithm that finds the optimal boundary between churners and non-churners using kernel functions.

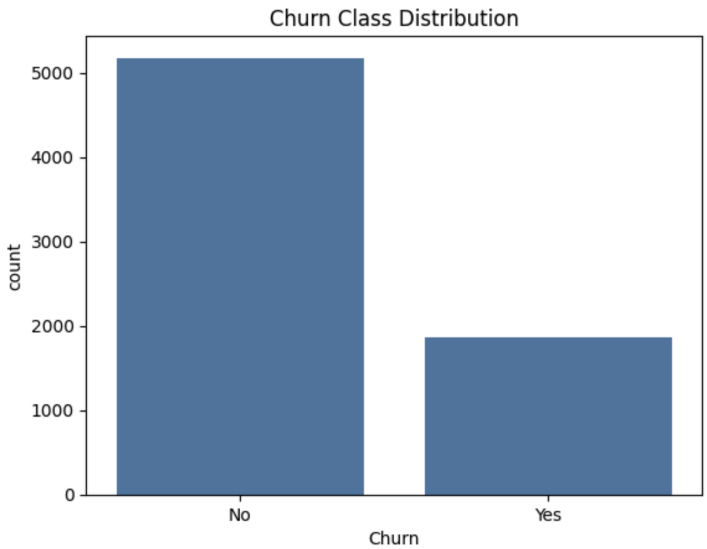
2. Dataset

Name: Telecommunication Churn Dataset - Kaggle
Link: https://www.kaggle.com/datasets/blastchar/telco-customer-churn

The Telco Customer Churn dataset contains information about customers from a telecommunications company, including demographic details, account information, and subscribed services.
Number of rows: 7,043
Number of features: 20 independent variables
Target variable: Churn - a binary variable indicating whether a customer has left the company (Yes = 1, No = 0).
Key attributes include customer tenure, contract type, payment method, monthly and total charges, and the presence of services such as internet, phone, and technical support.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
None
```

```
Unique values per column:
 customerID          7043
gender                 2
SeniorCitizen          2
Partner                2
Dependents             2
tenure                73
PhoneService           2
MultipleLines          3
InternetService        3
OnlineSecurity         3
OnlineBackup           3
DeviceProtection       3
TechSupport            3
StreamingTV            3
StreamingMovies        3
Contract               3
PaperlessBilling       2
PaymentMethod          4
MonthlyCharges      1585
TotalCharges        6531
Churn                  2
dtype: int64
```



Preprocessing Steps

Handling Missing Values: The TotalCharges column contained blank strings, which were converted to numeric values using pd.to_numeric(errors='coerce') and filled with the median. All other columns had no missing values.

Encoding Categorical Variables: Categorical features (e.g., Gender, Contract, PaymentMethod) were transformed using One-Hot Encoding to convert text labels into numerical format suitable for model training.

Normalization / Standardization: Numerical features such as tenure, MonthlyCharges, and TotalCharges were standardized using StandardScaler to ensure all variables were on comparable scales, improving model convergence.

Class Balance Handling: The dataset was slightly imbalanced (~26.5% churners vs. 73.5% non-churners). To address this, models were trained using class weights = ‘balanced’ (for Random Forest and SVM) and weighted loss (for Conv1D), ensuring fair learning from both classes.

3. Algorithms Overview

1. Random Forest

Rationale:

Random Forest was chosen as the baseline model because it is robust, handles both numerical and categorical features well, and performs automatic feature selection. It is also resistant to overfitting and provides strong predictive performance even with minimal parameter tuning.

How it works:

Random Forest is an ensemble learning algorithm that builds multiple independent decision trees on random subsets of data and features. Each tree votes for a class (churn or not), and the final prediction is made by majority vote. Averaging across many trees reduces variance and improves generalization.

2. Deep Learning Model – Conv1D

Rationale:

A 1D Convolutional Neural Network (Conv1D) was selected as the deep learning model because it can effectively capture local feature patterns and interactions within tabular data. Unlike dense networks, Conv1D applies convolution filters across the feature space, detecting meaningful combinations (e.g., between tenure, billing, and service types) that influence churn behavior.

Why it suits this task:

Handles complex nonlinear relationships among features.
Learns high-level representations automatically, reducing reliance on manual feature engineering.
Works efficiently on relatively small datasets compared to deeper architectures like LSTM.

3. Classic Machine Learning Model – Support Vector Machine (SVM)

Rationale:

SVM with an RBF (Radial Basis Function) kernel was selected as the traditional ML algorithm for its ability to model nonlinear decision boundaries effectively.

Strengths:

Performs well in high-dimensional feature spaces.
Maximizes the margin between classes, improving generalization.
Works effectively with smaller datasets.

Limitations:

Computationally expensive on large datasets.
Requires careful parameter tuning (C and gamma).

4. Implementation

Programming language: Python.

Development environment: Jupyter Notebook and .py scripts.

In VsCode

1. In your project folder, open a terminal and run:

python -m venv venv

venv\Scripts\activate # for Windows

pip install numpy pandas scikit-learn tensorflow keras matplotlib seaborn

OR

pip install requirements.txt (from my Github link)

2. To run standalone code:

python churn_models.py

3. To run Jupyter notebook:

jupyter notebook .\Yash_Shah_Final.ipynb

Dataset Loading Screenshot:

```
df = pd.read_csv(CSV_PATH)

print(df.head())
```

```
   customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No
1  5575-GNVDE    Male              0      No         No      34          Yes
2  3668-QPYBK    Male              0      No         No       2          Yes
3  7795-CFOCW    Male              0      No         No      45           No
4  9237-HQITU  Female              0      No         No       2          Yes

      MultipleLines InternetService OnlineSecurity  ... DeviceProtection  \
0  No phone service             DSL             No  ...               No
1                No             DSL            Yes  ...              Yes
2                No             DSL            Yes  ...               No
3  No phone service             DSL            Yes  ...              Yes
4                No     Fiber optic             No  ...               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

               PaymentMethod MonthlyCharges  TotalCharges Churn
0           Electronic check          29.85         29.85    No
1               Mailed check          56.95        1889.5    No
2               Mailed check          53.85        108.15   Yes
3  Bank transfer (automatic)          42.30       1840.75    No
4           Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
```

Training of algorithms:

```
[GLOBAL] RF best params: {'clf__max_depth': 20, 'clf__max_features': 'log2', 'clf__min_samples_leaf': 9, 'clf__min_samples_split': 8, 'clf__n_estimators': 400}
[GLOBAL] SVM best params: {'clf__C': np.float64(21.368329072358772), 'clf__gamma': np.float64(0.0007068974950624604)}
Epoch 1/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.6939 - val_loss: 0.6901 - learning_rate: 0.0010
Epoch 2/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 0.6778 - val_loss: 0.6883 - learning_rate: 0.0010
Epoch 3/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.6682 - val_loss: 0.6780 - learning_rate: 0.0010
Epoch 4/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.6370 - val_loss: 0.5724 - learning_rate: 0.0010
Epoch 5/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.6066 - val_loss: 0.6272 - learning_rate: 0.0010
Epoch 6/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.5813 - val_loss: 0.5547 - learning_rate: 0.0010
Epoch 7/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.5659 - val_loss: 0.6022 - learning_rate: 0.0010
Epoch 8/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5645 - val_loss: 0.5101 - learning_rate: 0.0010
Epoch 9/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5450 - val_loss: 0.5203 - learning_rate: 0.0010
Epoch 10/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5539 - val_loss: 0.5390 - learning_rate: 0.0010
Epoch 11/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5496 - val_loss: 0.5078 - learning_rate: 5.0000e-04
Epoch 12/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5520 - val_loss: 0.5273 - learning_rate: 5.0000e-04
Epoch 13/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.5311 - val_loss: 0.5388 - learning_rate: 5.0000e-04
Epoch 14/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5200 - val_loss: 0.5483 - learning_rate: 2.5000e-04
Epoch 15/36
```

Per fold Evaluation Output:

Is included in the Results section.

5. Evaluation Step

A 10-fold cross-validation approach was used to ensure robust model evaluation.
The dataset was split into 10 equal parts in each iteration, 9 folds were used for training and 1 fold for testing, rotating until every fold served as the test set once.
This reduces variance in performance estimates and provides a more reliable comparison across models.

Metrics Reported (per fold and overall averages):
Confusion-matrix metrics:
TP (True Positives), TN (True Negatives), FP (False Positives), FN (False Negatives)

Performance metrics:
Accuracy, Precision, Recall, F1-score

Rate-based metrics:
FPR (False Positive Rate), FNR (False Negative Rate), Specificity (TNR), Balanced Accuracy

Skill and reliability metrics:
TSS (True Skill Statistic), HSS (Heidke Skill Score)

Probability-based metrics:
ROC curve and AUC (Area Under the Curve)
BS (Brier Score) and BSS (Brier Skill Score)

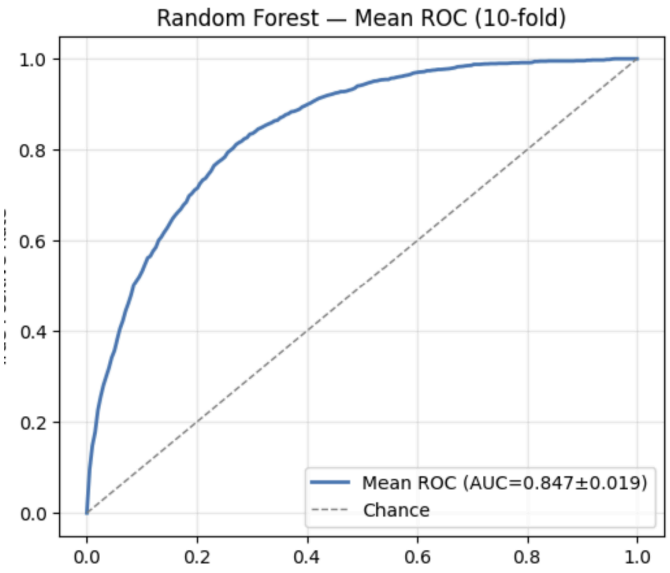
6. Results

Per fold (Random Forest) and ROC curve

```
=== Random Forest (per fold + average) ===
```

	Fold	TP	TN	FP	FN	P	N	TPR	TNR	FPR	...	Precision	Recall	F1	ErrorRate	TSS	HSS	BS	BSS	BSS_alt	AUC
0	1	146.0	402.0	116.0	41.0	187.0	518.0	0.7807	0.7761	0.2239	...	0.5573	0.7807	0.6503	0.2227	0.5568	0.4936	0.1515	0.2227	0.7773	0.8589
1	2	144.0	399.0	119.0	43.0	187.0	518.0	0.7701	0.7703	0.2297	...	0.5475	0.7701	0.6400	0.2298	0.5403	0.4782	0.1462	0.2499	0.7501	0.8651
2	3	147.0	388.0	130.0	40.0	187.0	518.0	0.7861	0.7490	0.2510	...	0.5307	0.7861	0.6336	0.2411	0.5351	0.4638	0.1578	0.1903	0.8097	0.8533
3	4	141.0	392.0	126.0	45.0	186.0	518.0	0.7581	0.7568	0.2432	...	0.5281	0.7581	0.6225	0.2429	0.5148	0.4518	0.1595	0.1797	0.8203	0.8362
4	5	148.0	406.0	111.0	39.0	187.0	517.0	0.7914	0.7853	0.2147	...	0.5714	0.7914	0.6637	0.2131	0.5767	0.5136	0.1458	0.2528	0.7472	0.8635
5	6	146.0	396.0	121.0	41.0	187.0	517.0	0.7807	0.7660	0.2340	...	0.5468	0.7807	0.6432	0.2301	0.5467	0.4810	0.1522	0.2199	0.7801	0.8574
6	7	138.0	377.0	140.0	49.0	187.0	517.0	0.7380	0.7292	0.2708	...	0.4964	0.7380	0.5935	0.2685	0.4672	0.4044	0.1809	0.0724	0.9276	0.7991
7	8	137.0	423.0	94.0	50.0	187.0	517.0	0.7326	0.8182	0.1818	...	0.5931	0.7326	0.6555	0.2045	0.5508	0.5123	0.1427	0.2687	0.7313	0.8562
8	9	147.0	381.0	136.0	40.0	187.0	517.0	0.7861	0.7369	0.2631	...	0.5194	0.7861	0.6255	0.2500	0.5230	0.4494	0.1603	0.1782	0.8218	0.8385
9	10	132.0	403.0	114.0	55.0	187.0	517.0	0.7059	0.7795	0.2205	...	0.5366	0.7059	0.6097	0.2401	0.4854	0.4410	0.1546	0.2073	0.7927	0.8381
10	Average	142.6	396.7	120.7	44.3	186.9	517.4	0.7630	0.7667	0.2333	...	0.5427	0.7630	0.6338	0.2343	0.5297	0.4689	0.1551	0.2042	0.7958	0.8466

11 rows × 23 columns

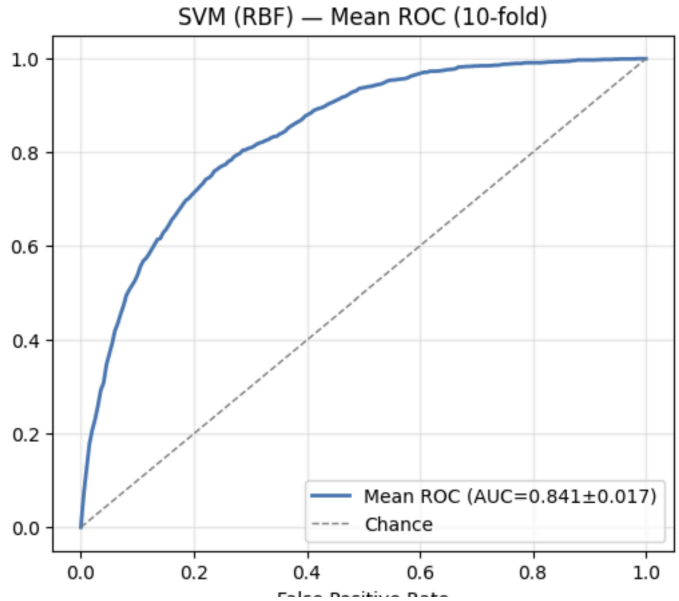


Per fold (SVM) and ROC curve

```
=== SVM (per fold + average) ===
```

	Fold	TP	TN	FP	FN	P	N	TPR	TNR	FPR	...	Precision	Recall	F1	ErrorRate	TSS	HSS	BS	BSS	BSS_alt	AUC
0	1	119.0	449.0	69.0	68.0	187.0	518.0	0.6364	0.8668	0.1332	...	0.6330	0.6364	0.6347	0.1943	0.5032	0.5023	0.1389	0.2872	0.7128	0.8566
1	2	79.0	490.0	28.0	108.0	187.0	518.0	0.4225	0.9459	0.0541	...	0.7383	0.4225	0.5374	0.1929	0.3684	0.4267	0.1559	0.1999	0.8001	0.8465
2	3	123.0	447.0	71.0	64.0	187.0	518.0	0.6578	0.8629	0.1371	...	0.6340	0.6578	0.6457	0.1915	0.5207	0.5145	0.1470	0.2457	0.7543	0.8559
3	4	107.0	460.0	58.0	79.0	186.0	518.0	0.5753	0.8880	0.1120	...	0.6485	0.5753	0.6097	0.1946	0.4633	0.4807	0.1583	0.1857	0.8143	0.8291
4	5	0.0	517.0	0.0	187.0	187.0	517.0	0.0000	1.0000	0.0000	...	0.0000	0.0000	0.0000	0.2656	0.0000	0.0000	0.1527	0.2172	0.7828	0.8649
5	6	67.0	500.0	17.0	120.0	187.0	517.0	0.3583	0.9671	0.0329	...	0.7976	0.3583	0.4945	0.1946	0.3254	0.3948	0.1557	0.2020	0.7980	0.8503
6	7	104.0	428.0	89.0	83.0	187.0	517.0	0.5561	0.8279	0.1721	...	0.5389	0.5561	0.5474	0.2443	0.3840	0.3801	0.1562	0.1991	0.8009	0.8027
7	8	71.0	491.0	26.0	116.0	187.0	517.0	0.3797	0.9497	0.0503	...	0.7320	0.3797	0.5000	0.2017	0.3294	0.3892	0.1560	0.2001	0.7999	0.8371
8	9	103.0	472.0	45.0	84.0	187.0	517.0	0.5508	0.9130	0.0870	...	0.6959	0.5508	0.6149	0.1832	0.4638	0.4968	0.1581	0.1896	0.8104	0.8314
9	10	98.0	453.0	64.0	89.0	187.0	517.0	0.5241	0.8762	0.1238	...	0.6049	0.5241	0.5616	0.2173	0.4003	0.4181	0.1541	0.2102	0.7898	0.8327
10	Average	87.1	470.7	46.7	99.8	186.9	517.4	0.4661	0.9098	0.0902	...	0.6023	0.4661	0.5146	0.2080	0.3758	0.4003	0.1533	0.2137	0.7863	0.8407

11 rows × 23 columns

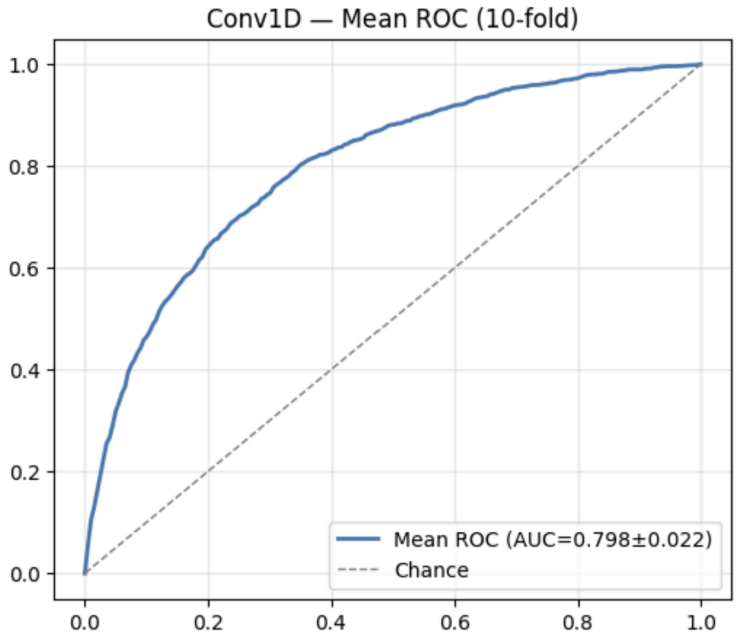


Per fold (Conv 1D) and ROC curve

```
=== Conv1D (per fold + average) ===
```

	Fold	TP	TN	FP	FN	P	N	TPR	TNR	FPR	...	Precision	Recall	F1	ErrorRate	TSS	HSS	BS	BSS	BSS_alt	AUC
0	1	131.0	417.0	101.0	56.0	187.0	518.0	0.7005	0.8050	0.1950	...	0.5647	0.7005	0.6253	0.2227	0.5056	0.4695	0.1655	0.1508	0.8492	0.8319
1	2	124.0	407.0	111.0	63.0	187.0	518.0	0.6631	0.7857	0.2143	...	0.5277	0.6631	0.5877	0.2468	0.4488	0.4148	0.1673	0.1418	0.8582	0.8195
2	3	128.0	405.0	113.0	59.0	187.0	518.0	0.6845	0.7819	0.2181	...	0.5311	0.6845	0.5981	0.2440	0.4663	0.4270	0.1865	0.0429	0.9571	0.8052
3	4	107.0	429.0	89.0	79.0	186.0	518.0	0.5753	0.8282	0.1718	...	0.5459	0.5753	0.5602	0.2386	0.4035	0.3966	0.1710	0.1203	0.8797	0.8026
4	5	121.0	443.0	74.0	66.0	187.0	517.0	0.6471	0.8569	0.1431	...	0.6205	0.6471	0.6335	0.1989	0.5039	0.4971	0.1792	0.0815	0.9185	0.8178
5	6	112.0	423.0	94.0	75.0	187.0	517.0	0.5989	0.8182	0.1818	...	0.5437	0.5989	0.5700	0.2401	0.4171	0.4040	0.1916	0.0180	0.9820	0.7743
6	7	120.0	376.0	141.0	67.0	187.0	517.0	0.6417	0.7273	0.2727	...	0.4598	0.6417	0.5357	0.2955	0.3690	0.3276	0.2001	-0.0259	1.0259	0.7596
7	8	98.0	457.0	60.0	89.0	187.0	517.0	0.5241	0.8839	0.1161	...	0.6203	0.5241	0.5681	0.2116	0.4080	0.4293	0.1750	0.1031	0.8969	0.7761
8	9	94.0	442.0	75.0	93.0	187.0	517.0	0.5027	0.8549	0.1451	...	0.5562	0.5027	0.5281	0.2386	0.3576	0.3689	0.1662	0.1477	0.8523	0.7890
9	10	105.0	428.0	89.0	82.0	187.0	517.0	0.5615	0.8279	0.1721	...	0.5412	0.5615	0.5512	0.2429	0.3894	0.3848	0.1741	0.1077	0.8923	0.8032
10	Average	114.0	422.7	94.7	72.9	186.9	517.4	0.6099	0.8170	0.1830	...	0.5511	0.6099	0.5758	0.2380	0.4269	0.4120	0.1776	0.0888	0.9112	0.7979

11 rows × 23 columns



Random Forest demonstrated the best overall balance between accuracy and recall, achieving an average accuracy of ~78.9%, F1 = 0.63, and AUC = 0.85. It maintained stable performance across folds with moderate variance and strong calibration (Brier Score ≈ 0.155).

SVM (RBF kernel) achieved the highest precision (0.60) and specificity (TNR ≈ 0.91), indicating it was more conservative and produced fewer false positives. However, it suffered from lower recall (0.47) and F1 (0.51), meaning it missed more true churners. Despite this, SVM's AUC (0.84) shows it still distinguished the two classes well.

Conv1D (Deep Learning) achieved balanced sensitivity and specificity (TPR ≈ 0.61, TNR ≈ 0.82) with an AUC of 0.80. It captured complex feature patterns but was slightly less stable across folds, possibly due to limited data size and training variability. Its Brier Skill Score (BSS ≈ 0.09) and F1 (0.57) indicate moderate predictive reliability.

7. Discussion

Algorithm Performance:

Among the three models, the Random Forest performed best overall, achieving the highest accuracy (≈78.9%), F1-score (0.63), and AUC (0.85). Its ensemble structure of multiple decision trees helped reduce variance and overfitting while effectively handling both numerical and categorical variables. The model's ability to capture nonlinear feature interactions made it particularly well-suited for the Telco Churn dataset, which contains a mix of demographic and service-based attributes. The SVM (RBF kernel) showed strong precision and specificity but lower recall, suggesting it was more conservative and better at identifying non-churners but missed some true churn cases. The Conv1D neural network performed reasonably well but did not surpass Random Forest, likely due to the dataset's small size which limited the benefits of deep learning architectures that usually require larger training sets to generalize effectively.

Challenges and Solutions:

A major challenge was handling the mixed feature types (categorical and numerical) and missing values in TotalCharges. This was addressed by using One-Hot Encoding for categorical variables, StandardScaler for numerical ones, and imputing missing values with the median. The dataset was also slightly imbalanced (≈26% churners), so class weights were applied to all models and weighted loss for the Conv1D model to ensure fair learning. Another challenge was training time, especially with 10-fold cross-validation and hyperparameter tuning. This was mitigated by performing a single global RandomizedSearchCV (3-fold) before cross-validation to reuse optimized parameters efficiently.

Potential Improvements and Future Work:

Future work could focus on improving recall (churn detection) using SMOTE or ADASYN oversampling, threshold tuning. For deep learning, experimenting with LSTM or Bi-LSTM architectures, dropout optimization, or hyperparameter tuning (via grid or Bayesian search) could yield better results. Additionally, applying explainability techniques such as SHAP or LIME would help interpret feature importance and improve trust in model predictions.

8. Conclusion

This project successfully implemented and compared three binary classification algorithms: Random Forest, Conv1D Neural Network, and Support Vector Machine (SVM) to predict customer churn using the Telco Customer Churn dataset. The Random Forest consistently outperformed the other models across most metrics, achieving the highest accuracy (≈79%), F1-score (0.63), and AUC (0.85), demonstrating strong generalization and robustness. The SVM achieved high precision and specificity, while Conv1D provided balanced performance but required more data to reach its full potential.

The findings highlight the impact of algorithm selection and evaluation methodology in predictive modeling. The use of 10-fold cross-validation ensured reliable and unbiased performance estimation, while comprehensive metrics such as AUC, F1, Brier Score, and Skill Scores provided a holistic view of model performance. Random Forest's ensemble nature offered the best trade-off between bias and variance for this mixed-type, moderately sized dataset.

Overall, the project demonstrates that ensemble models like Random Forest are well-suited for structured tabular data with both categorical and numerical features, whereas deep learning architectures may require larger datasets and more extensive tuning to outperform classical approaches. Future work can focus on feature engineering, data balancing, and model explainability to further improve churn prediction accuracy and interpretability.

9. References

Dataset Link: https://www.kaggle.com/datasets/blastchar/telco-customer-churn

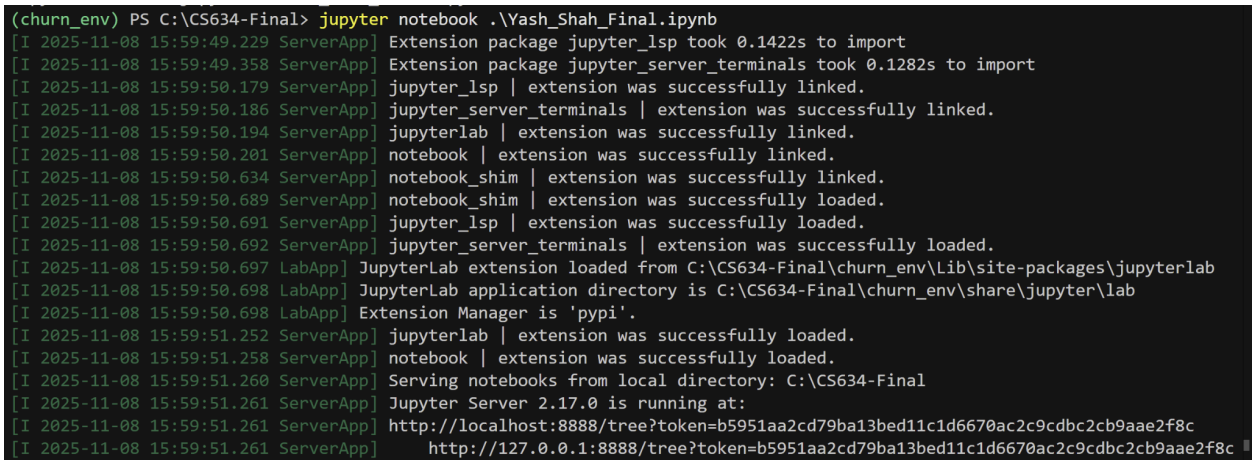
Referred to Class Slides.

10. Github Link

My personal account: https://github.com/yashah9/shah_yash_finaltermproj

11. Appendix

Screenshot of Jupyter Notebook (training and evaluation):



```
[GLOBAL] RF best params: {'clf__max_depth': 20, 'clf__max_features': 'log2', 'clf__min_samples_leaf': 9, 'clf__min_samples_split': 8, 'clf__n_estimators': 400}
[GLOBAL] SVM best params: {'clf__C': np.float64(21.368329072358772), 'clf__gamma': np.float64(0.0007068974950624604)}
Epoch 1/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.6939 - val_loss: 0.6901 - learning_rate: 0.0010
Epoch 2/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 0.6778 - val_loss: 0.6883 - learning_rate: 0.0010
Epoch 3/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.6682 - val_loss: 0.6780 - learning_rate: 0.0010
Epoch 4/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 0.6370 - val_loss: 0.5724 - learning_rate: 0.0010
Epoch 5/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.6066 - val_loss: 0.6272 - learning_rate: 0.0010
Epoch 6/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.5813 - val_loss: 0.5547 - learning_rate: 0.0010
Epoch 7/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.5659 - val_loss: 0.6022 - learning_rate: 0.0010
Epoch 8/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5645 - val_loss: 0.5101 - learning_rate: 0.0010
Epoch 9/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5450 - val_loss: 0.5203 - learning_rate: 0.0010
Epoch 10/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5539 - val_loss: 0.5390 - learning_rate: 0.0010
Epoch 11/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5496 - val_loss: 0.5078 - learning_rate: 5.0000e-04
Epoch 12/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5520 - val_loss: 0.5273 - learning_rate: 5.0000e-04
Epoch 13/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.5311 - val_loss: 0.5388 - learning_rate: 5.0000e-04
Epoch 14/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5200 - val_loss: 0.5483 - learning_rate: 2.5000e-04
Epoch 15/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5348 - val_loss: 0.5348 - learning_rate: 2.5000e-04
Epoch 1/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.6881 - val_loss: 0.6872 - learning_rate: 0.0010
Epoch 2/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - loss: 0.6852 - val_loss: 0.6855 - learning_rate: 0.0010
Epoch 3/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.6910 - val_loss: 0.6384 - learning_rate: 0.0010
Epoch 4/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.6656 - val_loss: 0.5793 - learning_rate: 0.0010
Epoch 5/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.6237 - val_loss: 0.5274 - learning_rate: 0.0010
Epoch 6/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5893 - val_loss: 0.5387 - learning_rate: 0.0010
Epoch 7/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5654 - val_loss: 0.5148 - learning_rate: 0.0010
Epoch 8/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5652 - val_loss: 0.5499 - learning_rate: 0.0010
Epoch 9/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5601 - val_loss: 0.5519 - learning_rate: 0.0010
Epoch 10/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5617 - val_loss: 0.5072 - learning_rate: 5.0000e-04
Epoch 11/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5395 - val_loss: 0.5279 - learning_rate: 5.0000e-04
Epoch 12/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5453 - val_loss: 0.5077 - learning_rate: 5.0000e-04
Epoch 13/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.5377 - val_loss: 0.5030 - learning_rate: 2.5000e-04
```

Evaluations are shown in results.

Screenshot of .py files (training and evaluation):

```
(churn_env) PS C:\CS634-Final> python .\churn_models.py
2025-11-10 22:23:32.270906: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly diffe
rent numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the
environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-11-10 22:23:37.467084: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly diffe
rent numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the
environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
Shapes: (7043, 19) (7043,)
Positive rate (churn=1): 0.2653698707936959

=== Fold 1/10 ===
2025-11-10 22:23:47.739396: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use
 available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the app
ropriate compiler flags.
Epoch 1/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.6924 - val_loss: 0.7031 - learning_rate: 0.0010
Epoch 2/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6888 - val_loss: 0.7050 - learning_rate: 0.0010
Epoch 3/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6733 - val_loss: 0.7268 - learning_rate: 0.0010
Epoch 4/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6490 - val_loss: 0.6564 - learning_rate: 5.0000e-04
Epoch 5/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6310 - val_loss: 0.5880 - learning_rate: 5.0000e-04
Epoch 6/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6155 - val_loss: 0.5999 - learning_rate: 5.0000e-04
Epoch 7/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.6054 - val_loss: 0.5792 - learning_rate: 5.0000e-04
Epoch 8/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.5993 - val_loss: 0.6183 - learning_rate: 5.0000e-04
Epoch 9/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.5958 - val_loss: 0.5586 - learning_rate: 5.0000e-04
Epoch 10/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.5886 - val_loss: 0.5683 - learning_rate: 5.0000e-04
Epoch 11/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.5793 - val_loss: 0.5465 - learning_rate: 5.0000e-04
Epoch 12/36
90/90 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.5769 - val_loss: 0.5832 - learning_rate: 5.0000e-04
```

```
(churn_env) PS C:\CS634-Final> python .\churn_models.py
       Fold  Accuracy  BalancedAcc  Precision  Recall      F1     TSS     HSS      BS     BSS     AUC
0         1    0.7773       0.7784     0.5573  0.7807  0.6503  0.5568  0.4936  0.1515  0.2227  0.8589
1         2    0.7702       0.7702     0.5475  0.7701  0.6400  0.5403  0.4782  0.1462  0.2499  0.8651
2         3    0.7589       0.7676     0.5307  0.7861  0.6336  0.5351  0.4638  0.1578  0.1903  0.8533
3         4    0.7571       0.7574     0.5281  0.7581  0.6225  0.5148  0.4518  0.1595  0.1797  0.8362
4         5    0.7869       0.7884     0.5714  0.7914  0.6637  0.5767  0.5136  0.1458  0.2528  0.8635
5         6    0.7699       0.7734     0.5468  0.7807  0.6432  0.5467  0.4810  0.1522  0.2199  0.8574
6         7    0.7315       0.7336     0.4964  0.7380  0.5935  0.4672  0.4044  0.1809  0.0724  0.7991
7         8    0.7955       0.7754     0.5931  0.7326  0.6555  0.5508  0.5123  0.1427  0.2687  0.8562
8         9    0.7500       0.7615     0.5194  0.7861  0.6255  0.5230  0.4494  0.1603  0.1782  0.8385
9        10    0.7599       0.7427     0.5366  0.7059  0.6097  0.4854  0.4410  0.1546  0.2073  0.8381
10  Average    0.7657       0.7648     0.5427  0.7630  0.6338  0.5297  0.4689  0.1551  0.2042  0.8466

=== SVM (10-fold) ===
       Fold  Accuracy  BalancedAcc  Precision  Recall      F1     TSS     HSS      BS     BSS     AUC
0         1    0.8057       0.7516     0.6330  0.6364  0.6347  0.5032  0.5023  0.1389  0.2872  0.8566
1         2    0.8071       0.6842     0.7383  0.4225  0.5374  0.3684  0.4267  0.1559  0.1999  0.8465
2         3    0.8085       0.7603     0.6340  0.6578  0.6457  0.5207  0.5145  0.1470  0.2457  0.8559
3         4    0.8054       0.7316     0.6485  0.5753  0.6097  0.4633  0.4807  0.1583  0.1857  0.8291
4         5    0.7344       0.5000     0.0000  0.0000  0.0000  0.0000  0.0000  0.1527  0.2172  0.8649
5         6    0.8054       0.6627     0.7976  0.3583  0.4945  0.3254  0.3948  0.1557  0.2020  0.8503
6         7    0.7557       0.6920     0.5389  0.5561  0.5474  0.3840  0.3801  0.1562  0.1991  0.8027
7         8    0.7983       0.6647     0.7320  0.3797  0.5000  0.3294  0.3892  0.1560  0.2001  0.8371
8         9    0.8168       0.7319     0.6959  0.5508  0.6149  0.4638  0.4968  0.1581  0.1896  0.8314
9        10    0.7827       0.7001     0.6049  0.5241  0.5616  0.4003  0.4181  0.1541  0.2102  0.8327
10  Average    0.7920       0.6879     0.6023  0.4661  0.5146  0.3758  0.4003  0.1533  0.2137  0.8407

=== Conv1D (10-fold) ===
       Fold  Accuracy  BalancedAcc  Precision  Recall      F1     TSS     HSS      BS     BSS     AUC
0         1    0.7844       0.7491     0.5806  0.6738  0.6238  0.4981  0.4738  0.1695  0.1305  0.8255
1         2    0.7872       0.7288     0.5979  0.6043  0.6011  0.4576  0.4560  0.1624  0.1666  0.8183
2         3    0.7447       0.7306     0.5137  0.7005  0.5928  0.4612  0.4132  0.1856  0.0475  0.8114
3         4    0.7628       0.6872     0.5537  0.5269  0.5399  0.3744  0.3803  0.1751  0.0991  0.7602
4         5    0.7997       0.7510     0.6173  0.6471  0.6319  0.5020  0.4944  0.1518  0.2220  0.8452
5         6    0.7713       0.7282     0.5613  0.6364  0.5965  0.4565  0.4378  0.1679  0.1394  0.8222
6         7    0.7003       0.6560     0.4487  0.5615  0.4988  0.3120  0.2888  0.2013 -0.0322  0.7366
7         8    0.7827       0.6848     0.6181  0.4759  0.5378  0.3696  0.3988  0.1603  0.1782  0.8034
8         9    0.7713       0.7180     0.5650  0.6043  0.5840  0.4360  0.4265  0.1675  0.1415  0.8063
9        10    0.7500       0.7206     0.5234  0.6578  0.5829  0.4411  0.4077  0.1721  0.1178  0.8137
10  Average    0.7654       0.7154     0.5580  0.6088  0.5789  0.4308  0.4177  0.1713  0.1210  0.8043

=== Summary (Averages Across Folds) ===
          Model  Accuracy  BalancedAcc  Precision  Recall      F1     TSS     HSS      BS     BSS     AUC
0  RandomForest    0.7657       0.7648     0.5427  0.7630  0.6338  0.5297  0.4689  0.1551  0.2042  0.8466
1       SVM_RBF    0.7920       0.6879     0.6023  0.4661  0.5146  0.3758  0.4003  0.1533  0.2137  0.8407
2        Conv1D    0.7654       0.7154     0.5580  0.6088  0.5789  0.4308  0.4177  0.1713  0.1210  0.8043
```

Additional Plots:

I have only built plots in the Jupyter Notebook and not in .py files because I was having trouble displaying them.

ROC curves are displayed in the Results section.

```
Encoded feature matrix shape: (7043, 45)
Example columns: ['num__SeniorCitizen', 'num__tenure', 'num__MonthlyCharges', 'num__TotalCharges', 'cat__gender_Female', 'cat__gender_Male', 'cat__Partner_No', 'cat__Partner_Yes', 'cat__Dependents_No', 'cat__Dependents_Yes']
```