CS634 Final Term Project Report

Student Name: Amanda Yu

Course: CS634 – Data Mining

Instructor: Dr. Yasser Abduallah

Project Title: Binary Classification of Income Data Using Random Forest, SVM, and CNN

Nov 16 2025

1. Introduction

My project implements multiple machine learning classification algorithms, with a deep learning model, to predict whether a person earns more than $50K per year. The prediction is based on various demographic and work related features provided in the dataset.

1.2 What Binary Classification Is and Why It's Important

Binary classification is a machine learning task where the goal is to sort data into two categories, such like "yes or no" or "1 or 0". The model learns from labeled examples so it can predict what two groups a new sample belongs to.

This is important because many problems in the real world involve two outcomes, like detecting fraud, identifying diseases, filtering spam, or, in my project, predicting whether someone earns more or less than $50K. Binary classification provides a simple but powerful way to make reliable decisions in these situations.

1.3 Dataset and Algorithms Used

This project uses the Adult Income Dataset found on Kaggle

This project implements machine learning algorithms, including:

- Random Forest

- Support Vector Machine (SVM)

- A 1D Convolutional Neural Network (CNN) for deep learning

2. Dataset

2.1 Name & Source

American Citizens Annual Income on Kaggle

link: https://www.kaggle.com/datasets/amirhosseinmirzaie/americancitizenincome/code

2.2 Description:

The original dataset has 25,000 rows and 15 features, covering demographic and work related attributes such as age, education, occupation, capital gain/loss, and working hours weekly. The target variable is income, if a person earns more than $50K or not.

Because of this project requirement, it was trimmed down to 224 entries while retaining the same 15 features and 26 KB

2.3 Preprocessing steps:

Required packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import gaussian_kde
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, brier_score_loss
```

Preprocessing steps: removed missing values, normalization/standardization, label encoding, class balance handling

3. Algorithms Overview

Random Forest

An ensemble learning technique that creates and combines the predictions of decision trees.

Each tree is constructed by randomly sampling from both the data and the possible data features which typically helps reduce overfitting and improve generalisation. It is effective for tabular data, can utilize either numerical or categorical data, and can find complex, non-linear structures. Random Forest was chosen to provide a reliable and straightforward option and often provides very good performance on structured data, such as the dataset considered here.

Support Vector Machines (SVMs)

This algorithm is a strong baseline classifier, especially for binary classification tasks. SVMs work by finding the best boundary or hyperplane between the two classes. Some strengths of SVM include better performance on smaller datasets, robustness to high dimensional data, and the ability to model complex boundaries using different kernels. However, they also can be slow with larger datasets, require careful parameter tuning, and are less interpretable than simple decision trees. Regardless, they still offer a solid comparison benchmark against both Random Forest and CNN.

CNNs

CNNs are usually used with images, a 1D version can learn local patterns and interactions between neighbouring features in tabular data. This gives it the ability to capture relationships that simpler models might miss. It also scales well when more data or features are added. A CNN is useful here because it can automatically learn feature representations and may uncover subtle income-related patterns that aren't obvious to traditional ML models.

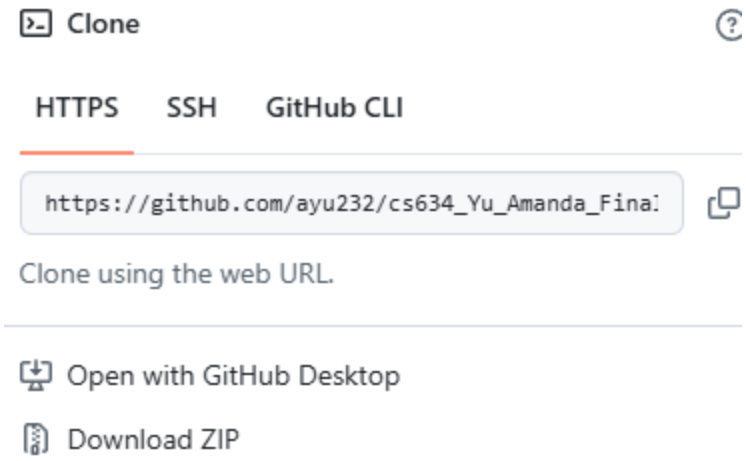
4. Implementation

Programming language: Python.

Development environment: Jupyter Notebook and .py scripts.

Steps to be taken for the Installation and Execution of the Project

1. Clone from github and click download zip



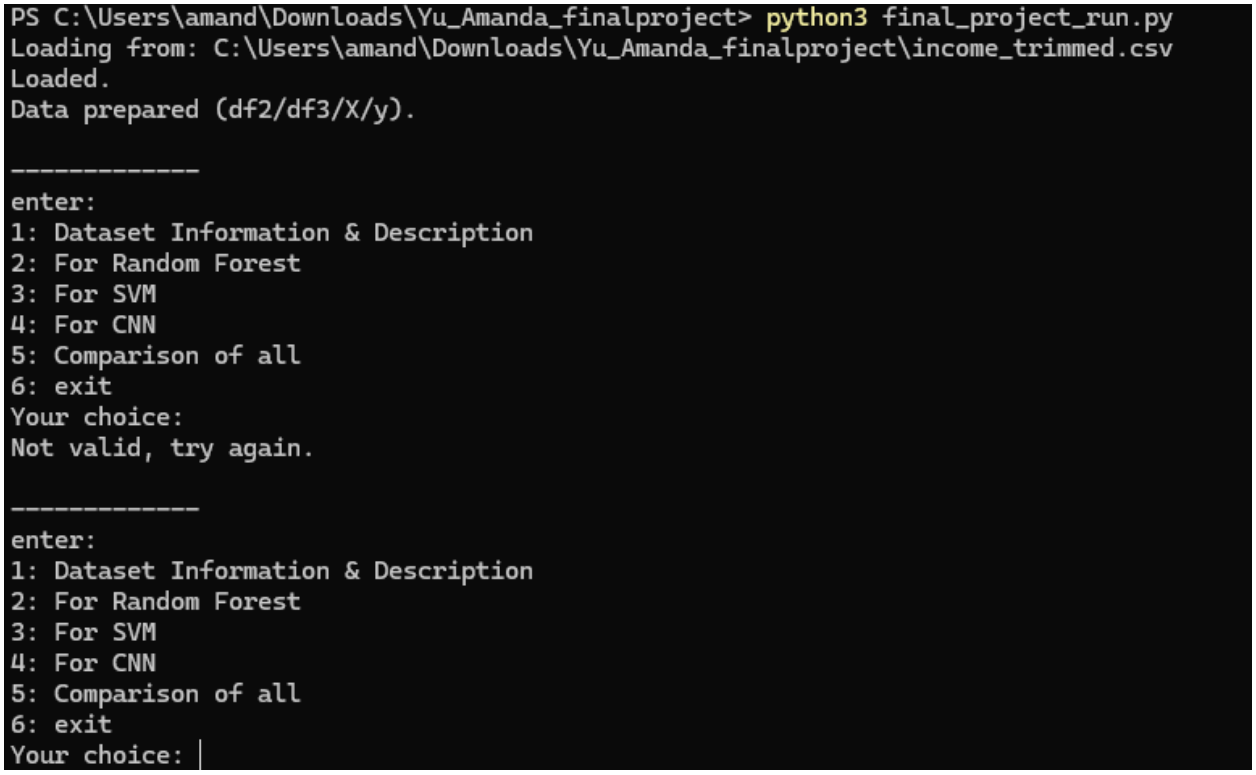
2. Cd to downloads then cd Yu_Amanda_finalproject or if cd Yu_Amanda_finalproject works right away use that

```
PS C:\Users\amand> cd Downloads
PS C:\Users\amand\Downloads> cd Yu_Amanda_finalproject
```

3**. pip install -r requirements.txt** to install all of required packages alternatively do **pip install pandas numpy matplotlib seaborn scipy scikit-learn torch**

```
PS C:\Users\amand\Downloads\Yu_Amanda_finalproject> pip install -r requirements.txt
ERROR: Invalid requirement: 'pandas numpy matplotlib seaborn scipy scikit-learn torch' (from line 1 of requirements.txt)

[notice] A new release of pip is available: 24.0 -> 25.3
[notice] To update, run: C:\Users\amand\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\python.exe -m pip install --upgrade pip
```

4. Run final_project_run.py with **python3 final_project_run.py**



5. Input 1 for : Dataset Information & Description

```
enter:
1: Dataset Information & Description
2: For Random Forest
3: For SVM
4: For CNN
5: Comparison of all
6: exit
Your choice: 1

*** Dataset Info ***

Raw df.info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 224 entries, 0 to 223
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             224 non-null    int64
 1   workclass       224 non-null    object
 2   fnlwgt          224 non-null    int64
 3   education       224 non-null    object
 4   education_num   224 non-null    int64
 5   marital_status  224 non-null    object
 6   occupation      224 non-null    object
 7   relationship    223 non-null    object
 8   race            223 non-null    object
 9   sex             223 non-null    object
 10  capital_gain    223 non-null    float64
 11  capital_loss    223 non-null    float64
 12  hours_per_week  223 non-null    float64
 13  native_country  223 non-null    object
 14  income          223 non-null    object
dtypes: float64(3), int64(3), object(9)
memory usage: 26.4+ KB
None
```

```
Head of df:
   age         workclass  fnlwgt     education  ...  capital_loss hours_per_week native_country income
0   40  Self-emp-not-inc  223881   Prof-school  ...           0.0           70.0  United-States   >50K
1   30           Private  149118       HS-grad  ...           0.0           40.0  United-States  <=50K
2   46           Private  109209  Some-college  ...           0.0           40.0  United-States   >50K
3   32           Private  229566     Assoc-voc  ...           0.0           60.0  United-States   >50K
4   54                 ?  148657     Preschool  ...           0.0           40.0         Mexico  <=50K
5   63           Private  111963  Some-college  ...           0.0           16.0  United-States  <=50K
6   25           Private  207875       7th-8th  ...           0.0           40.0         Mexico  <=50K
7   71         Local-gov  229110       HS-grad  ...           0.0           33.0  United-States  <=50K
8   37           Private   66686       HS-grad  ...           0.0           40.0  United-States  <=50K
9   44           Private  227399     Assoc-voc  ...           0.0           40.0  United-States  <=50K

[10 rows x 15 columns]

Cleaned df2 info:
<class 'pandas.core.frame.DataFrame'>
Index: 208 entries, 0 to 222
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             208 non-null    int64
 1   workclass       208 non-null    object
 2   fnlwgt          208 non-null    int64
 3   education       208 non-null    object
 4   education_num   208 non-null    int64
 5   marital_status  208 non-null    object
 6   occupation      208 non-null    object
 7   relationship    208 non-null    object
 8   race            208 non-null    object
 9   sex             208 non-null    object
 10  capital_gain    208 non-null    float64
 11  capital_loss    208 non-null    float64
 12  hours_per_week  208 non-null    float64
 13  native_country  208 non-null    object
```

6. Distribution of each numerical attribute will load exit to continue. Distribution of above and below $50K will also appear to continue click exit again. Then KDE plots exit all until no more plots show up

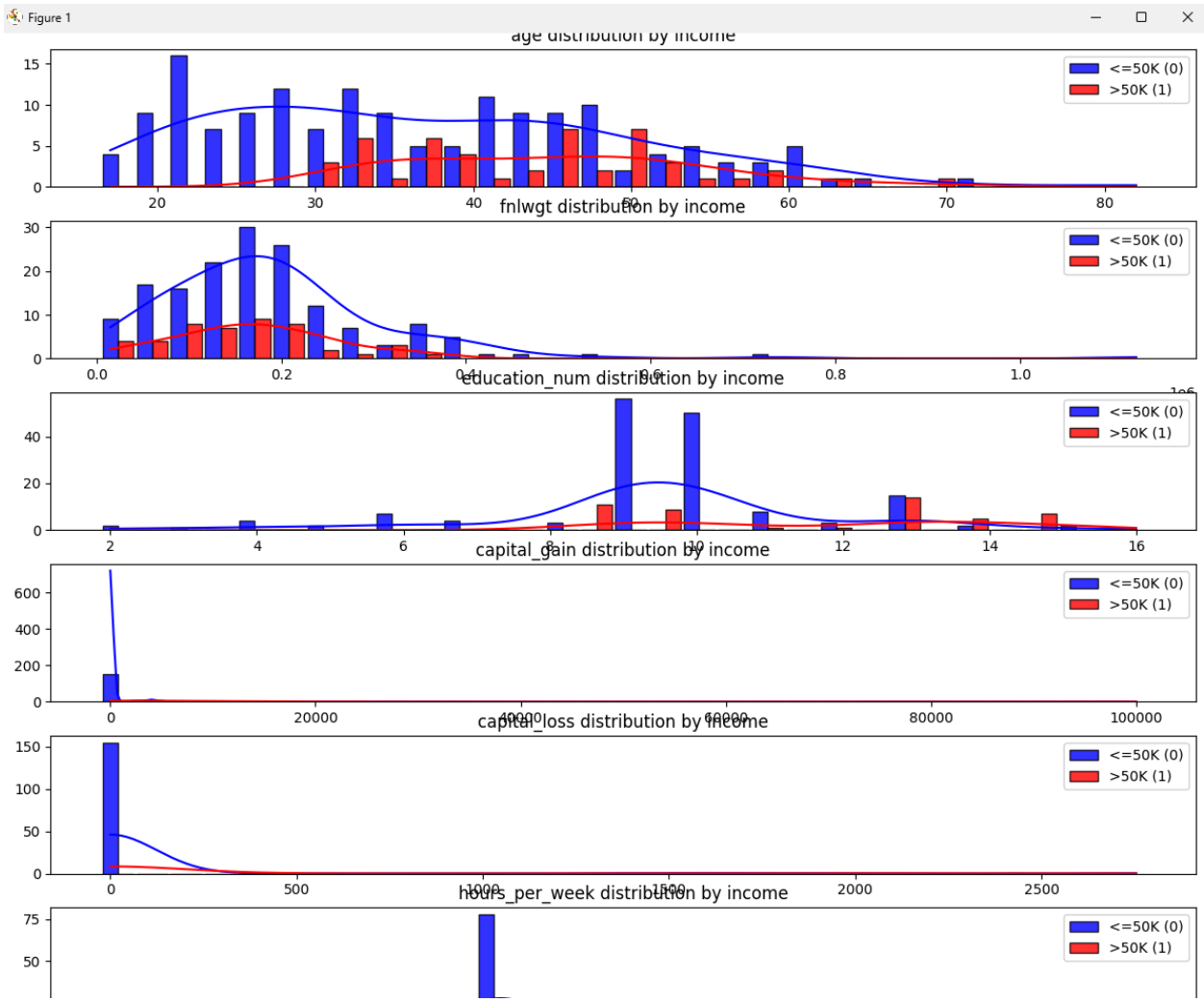
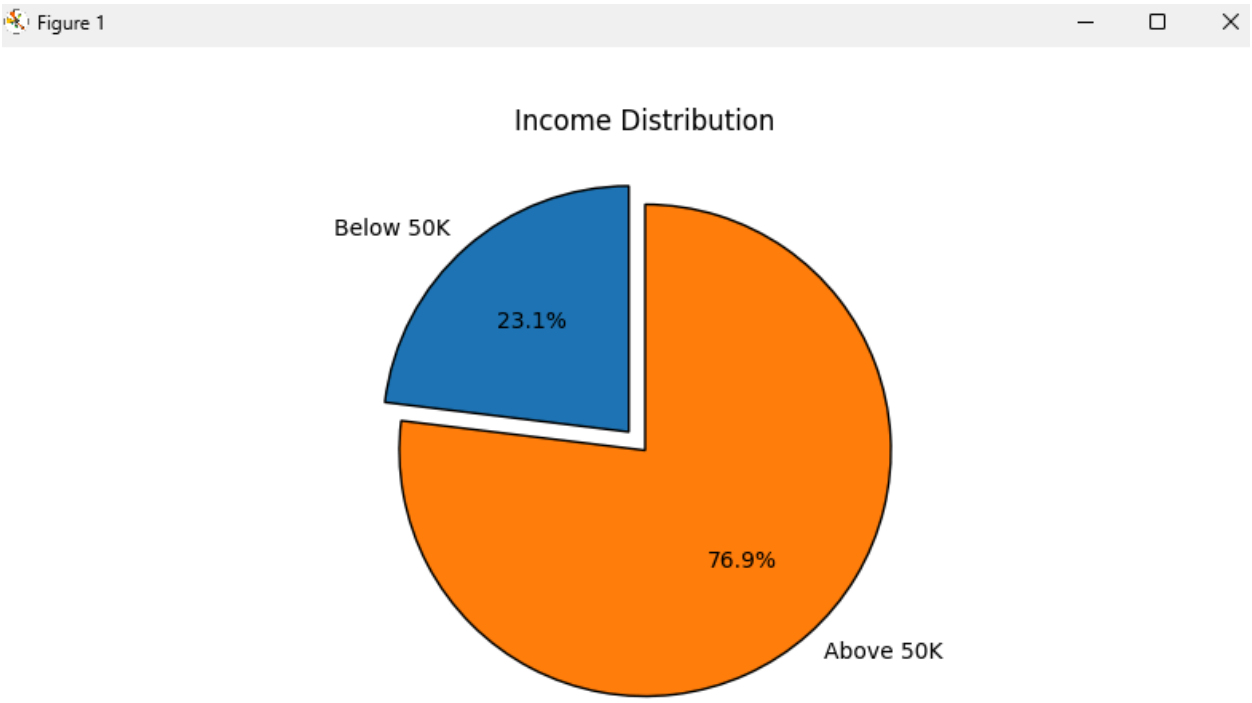
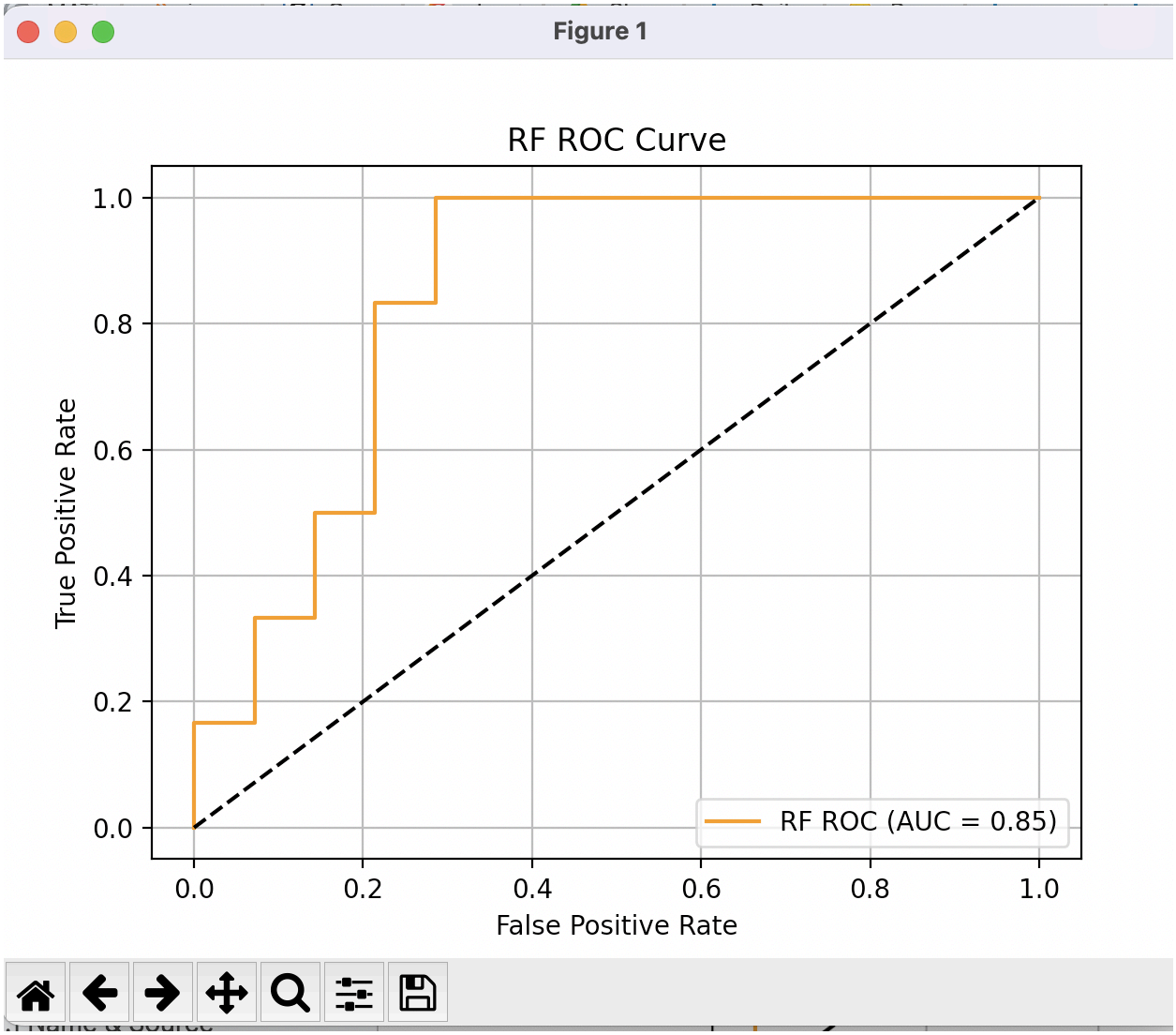


Figure 1
age distribution by income
<=50K (0)
>50K (1)
fnlwgt distribution by income
education_num distribution by income
capital_gain distribution by income
capital_loss distribution by income
hours_per_week distribution by income



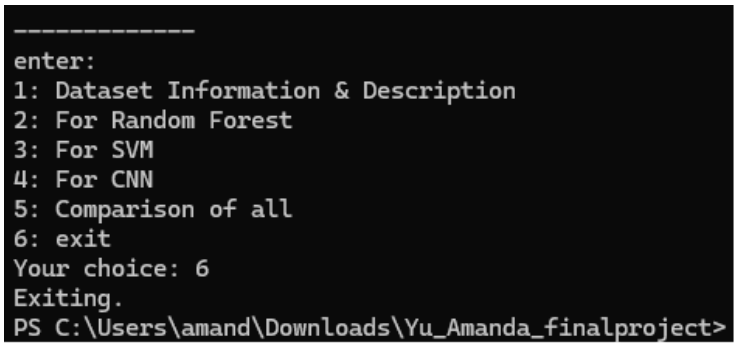
7. Input 2 for For Random Forest training and evaluation metrics for each iteration in 10 fold k-fold validation (note: first pass will train all models together) will also show ROC curve for

```
enter:
1: Dataset Information & Description
2: For Random Forest
3: For SVM
4: For CNN
5: Comparison of all
6: exit
Your choice: 2

*** Random Forest metrics across folds ***
                iter1      iter2      iter3      iter4      iter5      iter6      iter7      iter8      iter9     iter10
TP           3.000000   1.000000   1.000000   2.000000   4.000000   3.000000   3.000000   2.000000   2.000000   2.000000
TN          14.000000  17.000000  16.000000  16.000000  11.000000  15.000000  15.000000  16.000000  15.000000  13.000000
FP           2.000000   2.000000   0.000000   2.000000   0.000000   2.000000   0.000000   3.000000   0.000000   1.000000
FN           2.000000   1.000000   4.000000   1.000000   6.000000   1.000000   3.000000   0.000000   3.000000   4.000000
TPR          0.600000   0.500000   0.200000   0.666667   0.400000   0.750000   0.500000   1.000000   0.400000   0.333333
TNR          0.875000   0.894737   1.000000   0.888889   1.000000   0.882353   1.000000   0.842105   1.000000   0.928571
FPR          0.125000   0.105263   0.000000   0.111111   0.000000   0.117647   0.000000   0.157895   0.000000   0.071429
FNR          0.400000   0.500000   0.800000   0.333333   0.600000   0.250000   0.500000   0.000000   0.600000   0.666667
Precision    0.600000   0.333333   1.000000   0.500000   1.000000   0.600000   1.000000   0.400000   1.000000   0.666667
Accuracy     0.809524   0.857143   0.809524   0.857143   0.714286   0.857143   0.857143   0.857143   0.850000   0.750000
Recall       0.600000   0.500000   0.200000   0.666667   0.400000   0.750000   0.500000   1.000000   0.400000   0.333333
F1_measure   0.600000   0.400000   0.333333   0.571429   0.571429   0.666667   0.666667   0.571429   0.571429   0.444444
Error_rate   0.190476   0.142857   0.190476   0.142857   0.285714   0.142857   0.142857   0.142857   0.150000   0.250000
BAcc         0.737500   0.697368   0.600000   0.777778   0.700000   0.816176   0.750000   0.921053   0.700000   0.630952
TSS          0.475000   0.394737   0.200000   0.555556   0.400000   0.632353   0.500000   0.842105   0.400000   0.261905
HSS          0.475000   0.322581   0.275862   0.487805   0.411215   0.577181   0.588235   0.503937   0.500000   0.305556
AUC          0.900000   0.894737   0.900000   0.925926   0.945455   0.919118   0.905556   0.973684   0.873333   0.845238
Brier_score  0.113710   0.090133   0.118824   0.090833   0.174676   0.103319   0.112176   0.113238   0.125395   0.171775
BSS          0.373406   0.172282   0.345223   0.311925   0.460796   0.338562   0.460216  -0.039895   0.332837   0.204255

Avg RF metrics (over folds):
TP             2.300000
TN            14.800000
```



8. Repeat Input 3 - 5 to get SVM then CNN then all together (RF, SVM, and CNN)

9. Input 6 to exit



5. Evaluation Setup

In this project, I tested several different classification models, including a classic machine learning model, a Random Forest classifier, and a deep learning mode, to predict whether a person earns more than $50K per year. The models use demographic and work-related features from the dataset to learn patterns that separate high-income and lower-income individuals.

The model performed was evaluated using 10-fold cross-validation. The dataset is divided into ten equal portions, known as folds. In each round, one fold acts as a test set, while the remaining nine are used for training. This repeats ten times, with every data point used once for testing. Cross-validation provides a better estimate of performance than a single train-test split.

For each fold and for each algorithm, I kept track of a collection of evaluation metrics. These include the basic confusion matrix values:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)

From these, I calculated several performance measures:

- Accuracy
- Precision
- Recall
- F1-score

- False Positive Rate (FPR)
- False Negative Rate (FNR)
- Specificity (TNR)
- Balanced Accuracy

I also included additional skill-based metrics commonly used in classification tasks:

- True Skill Statistic (TSS)
- Heidke Skill Score (HSS)

Finally, I looked at probability and ranking-based metrics:

- ROC curve
- Area Under the Curve (AUC)
- Brier Score (BS)
- Brier Skill Score (BSS)

After computing all these metrics for each fold, I averaged the results so that each algorithm has an overall performance summary across the 10 folds.

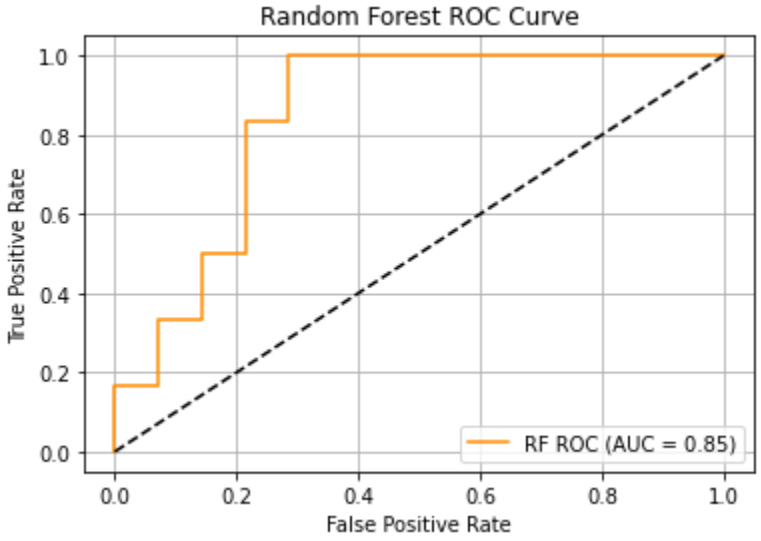
6. Results

Random forests

Iteration	iter1	iter2	iter3	iter4	iter5	iter6	iter7	iter8	iter9	iter10
TP	3	1	1	2	4	3	3	2	2	2
TN	14	17	16	16	11	15	15	16	15	13
FP	2	2	0	2	0	2	0	3	0	1
FN	2	1	4	1	6	1	3	0	3	4

TPR	0.6	0.5	0.2	0.666667	0.4	0.75	0.5	1	0.4	0.333333
TNR	0.875	0.894737	1	0.888889	1	0.882353	1	0.842105	1	0.928571
FPR	0.125	0.105263	0	0.111111	0	0.117647	0	0.157895	0	0.071429
FNR	0.4	0.5	0.8	0.333333	0.6	0.25	0.5	0	0.6	0.666667
Precision	0.6	0.333333	1	0.5	1	0.6	1	0.4	1	0.666667
Accuracy	0.809524	0.857143	0.809524	0.857143	0.714286	0.857143	0.857143	0.857143	0.85	0.75
Recall	0.6	0.5	0.2	0.666667	0.4	0.75	0.5	1	0.4	0.333333
F1_measure	0.6	0.4	0.333333	0.571429	0.571429	0.666667	0.666667	0.5714299	0.571429	0.444444
Error_rate	0.190476	0.142857	0.190476	0.142857	0.285714	0.142857	0.142857	0.142857	0.15	0.25
BAcc	0.7375	0.697368	0.6	0.777778	0.7	0.816176	0.75	0.921053	0.7	0.630952
TSS	0.475	0.394737	0.2	0.555556	0.4	0.632353	0.5	0.842105	0.4	0.261905
HSS	0.475	0.322581	0.275862	0.487805	0.411215	0.577181	0.588235	0.503937	0.5	0.305556
AUC	0.9	0.894737	0.9	0.925926	0.945455	0.919118	0.905556	0.9736843	0.873333	0.845238
Brier_score	0.11371	0.090133	0.118824	0.090833	0.174676	0.103319	0.112176	0.113238	0.125395	0.171775

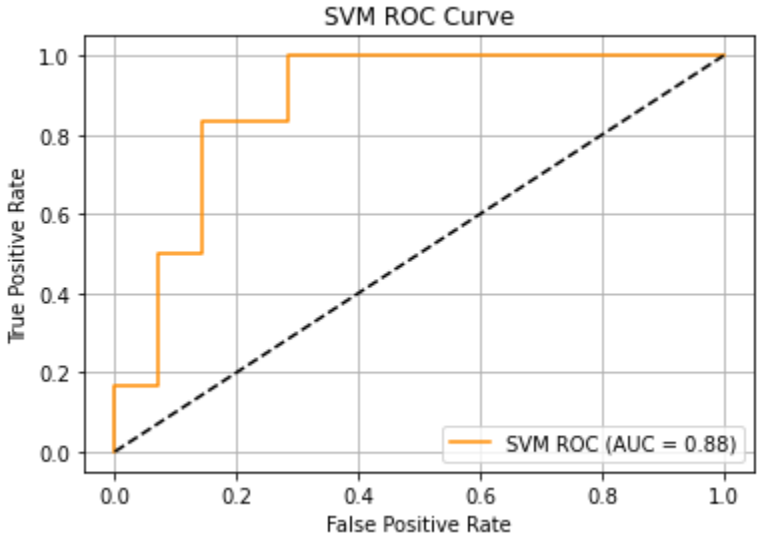
BSS	0.373406	0.172282	0.345223	0.311925	0.460796	0.338562	0.460216	-0.039895	0.332837	0.204255



SVM

Iteration	iter1	iter2	iter3	iter4	iter5	iter6	iter7	iter8	iter9	iter10
TP	5	1	3	3	8	3	5	2	3	4
TN	12	14	16	13	10	11	14	15	14	12
FP	4	5	0	5	1	6	1	4	1	2
FN	0	1	2	0	2	1	1	0	2	2
TPR	1	0.5	0.6	1	0.8	0.75	0.833333	1	0.6	0.666667
TNR	0.75	0.736842	1	0.722222	0.909091	0.647059	0.933333	0.789474	0.933333	0.857143
FPR	0.25	0.263158	0	0.277778	0.090909	0.352941	0.066667	0.210526	0.066667	0.142857
FNR	0	0.5	0.4	0	0.2	0.25	0.166667	0	0.4	0.333333
Precision	0.555556	0.166667	1	0.375	0.888889	0.333333	0.833333	0.333333	0.75	0.666667
Accuracy	0.809524	0.714286	0.904762	0.761905	0.857143	0.666667	0.904762	0.809524	0.85	0.8

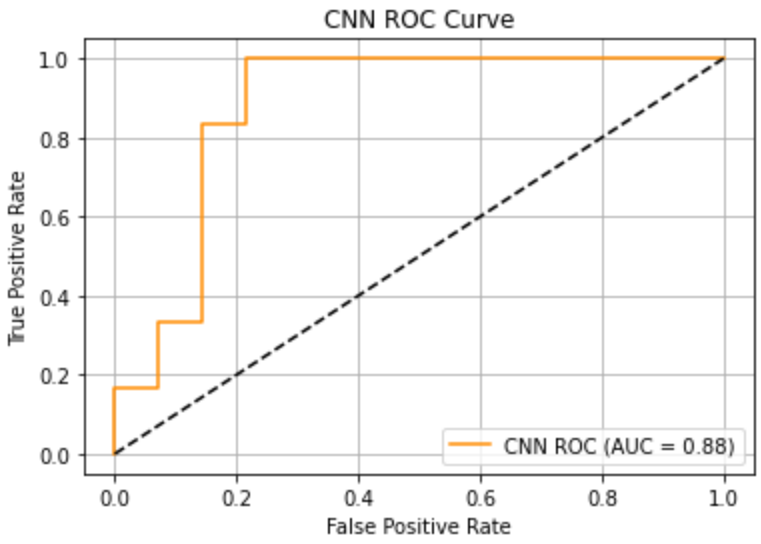
Recall	1	0.5	0.6	1	0.8	0.75	0.833333	1	0.6	0.666667
F1_measure	0.714286	0.25	0.75	0.545455	0.842105	0.461538	0.833333	0.5	0.666667	0.666667
Error_rate	0.190476	0.285714	0.095238	0.238095	0.142857	0.333333	0.095238	0.190476	0.15	0.2
BAcc	0.875	0.618421	0.8	0.861111	0.854545	0.698529	0.883333	0.894737	0.766667	0.761905
TSS	0.75	0.236842	0.6	0.722222	0.709091	0.397059	0.766667	0.789474	0.533333	0.52381
HSS	0.588235	0.125	0.695652	0.42623	0.712329	0.268657	0.766667	0.416667	0.571429	0.52381
AUC	0.8625	0.842105	0.875	0.962963	0.945455	0.882353	0.9	0.973684	0.84	0.880952
Brier_score	0.137904	0.099859	0.102306	0.088575	0.223207	0.092779	0.094369	0.075065	0.146563	0.148631
BSS	0.24008	0.082965	0.436244	0.329032	0.310989	0.406038	0.545904	0.310658	0.220214	0.311469



CNN

Iteration	iter1	iter2	iter3	iter4	iter5	iter6	iter7	iter8	iter9	iter10
TP	5	2	4	3	8	3	6	2	4	5

TN	12	17	15	13	10	13	13	15	12	12
FP	4	2	1	5	1	4	2	4	3	2
FN	0	0	1	0	2	1	0	0	1	1
TPR	1	1	0.8	1	0.8	0.75	1	1	0.8	0.833333
TNR	0.75	0.894737	0.9375	0.722222	0.909091	0.764706	0.866667	0.789474	0.8	0.857143
FPR	0.25	0.105263	0.0625	0.277778	0.090909	0.235294	0.133333	0.210526	0.2	0.142857
FNR	0	0	0.2	0	0.2	0.25	0	0	0.2	0.166667
Precision	0.555556	0.5	0.8	0.375	0.888889	0.428571	0.75	0.333333	0.571429	0.714286
Accuracy	0.809524	0.904762	0.904762	0.761905	0.857143	0.761905	0.904762	0.809524	0.8	0.85
Recall	1	1	0.8	1	0.8	0.75	1	1	0.8	0.833333
F1_measure	0.714286	0.666667	0.8	0.545455	0.842105	0.545455	0.857143	0.5	0.666667	0.769231
Error_rate	0.190476	0.095238	0.095238	0.238095	0.142857	0.238095	0.095238	0.190476	0.2	0.15
BAcc	0.875	0.947368	0.86875	0.861111	0.854545	0.757353	0.933333	0.894737	0.8	0.845238
TSS	0.75	0.894737	0.7375	0.722222	0.709091	0.514706	0.866667	0.789474	0.6	0.690476
HSS	0.588235	0.618182	0.7375	0.42623	0.712329	0.4	0.787879	0.416667	0.529412	0.659091
AUC	0.8875	1	0.9625	1	0.981818	0.926471	0.977778	0.973684	0.92	0.880952
Brier_score	0.15442	0.112072	0.071636	0.129673	0.085655	0.132541	0.072543	0.137431	0.108327	0.127293
BSS	0.149072	-0.029188	0.60525	0.017709	0.735594	0.151484	0.650929	-0.262066	0.423647	0.410315



7. Discussion

Which algorithm performed best overall and why?

CNN achieved the strongest overall performance, especially in terms of AUC, balanced accuracy, TPR, and F1-score. Although Random Forest and SVM also performed well in several folds, CNN was more consistent with high recall and ability to find patterns, even with a small dataset.

Why CNN performed best:

1. Low Bias / Higher Flexibility:

The CNN can model complex relationships between features because of its learned convolutional filters. Even in small datasets, it can discover feature interactions that linear models may miss.

2. Handling of Non-Linear Patterns better:

Income prediction involves interactions between age, education, hours worked, etc. CNNs can learn these interactions naturally.

3. The Dataset Is Small but Structured:

SVM and RF Limitations:

SVM performed strongly in recall but struggled in precision, showing high variance depending on the fold.

Random Forest performed well and consistently but showed more variance and lower recall in folds where the positive class was underrepresented.

CNN had the best trade off between recall and precision, and the highest AUC values, meaning it separated the classes most effectively.

Challenges

1. Missing Values
    a. Some rows had missing entries after trimming the dataset.
    b. Solution: Replace ? with NaN and drop rows with missing values. This ensured all models trained on clean, consistent data.
2. Extremely Small Dataset
    a. After trimming to 224 records, the dataset became prone to:high variance, unstable decision boundaries, and difficulty generalizing
    b. Solution: 10-fold cross-validation provided a more reliable performance estimate and maximized data usage.
3. Class Imbalance
    a. Income >50K is more common than <=50K. This caused models to predict the majority class too often.
    b. Solution: class_weight='balanced' for RF and SVM
        i. pos_weight in CNN's BCEWithLogitsLoss
        ii. These adjustments penalized mistakes on the minority class to improve recall.
4. Scaling Numerical Features.
    a. Algorithms like SVM and CNN need normalized data.
    b. Solution: StandardScaler was applied to numeric features.
5. CNN Training Stability
    a. mall dataset, CNNs can easily overfit.
    b. Solution: Shallow architecture, small number of epochs, and weight balancing all kept the weight balanced

Potential Improvements and Future Work

1. Dataset Expansion
    a. CNNs and Random Forests can benefit from more data.
    b. Larger datasets would reduce variance and allow deeper architectures.

2. Hyperparameter Tuning

a. Grid search or random search could optimize: number of trees (RF), kernel/C parameter (SVM), convolution filters, learning rate, batch size (CNN). This would most likely improve performance for all models.

3. Feature Engineering

a. Creating additional features could improve predictive power: Interaction features (e.g., age × education) and Encoding occupation or country into meaningful categories

4. Class Rebalancing

a. Instead of dropping missing rows, I could have: Use SMOTE for synthetic oversampling, tried undersampling the majority class. These could potentially boost recall for the imbalanced class

6. Improve CNN Architecture

a. Add dropout for regularization, add more filters or another convolution layer or even, add batch normalization

8. Conclusion

Summary of Key Findings and Takeaways

When we ran the 10-fold cross-validation, all three models did a good job classifying income, there were some tradeoffs however. CNN stood out since it had the highest average true positives (3.9) and kept false negatives to a minimum (0.9). In plain terms, it caught more of the high earners (>50K), which was impressive since they only made up 23.1% of the data.

SVM did well too, especially at finding the >50K group, but it produced more false positives than the CNN. Random Forest went the other way: it achieved the most true negatives (14.8 on average),but also the highest false negatives, meaning it tended to underpredict the >50K class. This reflects its more conservative bias when handling imbalanced datasets.

CNN handled things the most evenly, while the other models leaned either toward precision (Random Forest) or recall (SVM). These differences really show how the model's design and the class imbalance can shape what you get out of them.

Reflection on Algorithm Choice and Evaluation Methodology

The choice of algorithm had a clear impact on performance. Random Forest was stable and

interpretable, but it struggled with the minority >50K class because the dataset was heavily imbalanced. SVM handled the imbalance better through class weighting but showed higher variance across folds due to the small dataset size. CNN, even with only 224 samples, benefited from its ability to learn feature interactions and generalize patterns without relying on manual feature engineering.

Another key factor was the methodology of evaluation, given the extremely small size of the dataset, a simple train/test split would not have been reliable. 10-fold cross-validation allowed every data point to be used for training and testing, reducing variance and giving a more trustworthy estimate of performance. It also let us see the stability or instability of each model across folds, something which a single split would have entirely missed.

The strong performance of the CNN shows that even simple deep learning architectures can outperform classic models on structured data, provided the evaluation is rigorous and class imbalance is addressed properly.

9. References

Dataset source link:
https://www.kaggle.com/datasets/amirhosseinmirzaie/americancitizenincome/code

Followed Abid_mahum_finalproject for some preprocessing steps

Random Forest tutorial: https://www.datacamp.com/tutorial/random-forests-classifier-python

SVM tutorial: https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python

CNN tutorial:
https://coderzcolumn.com/tutorials/artificial-intelligence/pytorch-conv1d-for-text-classification

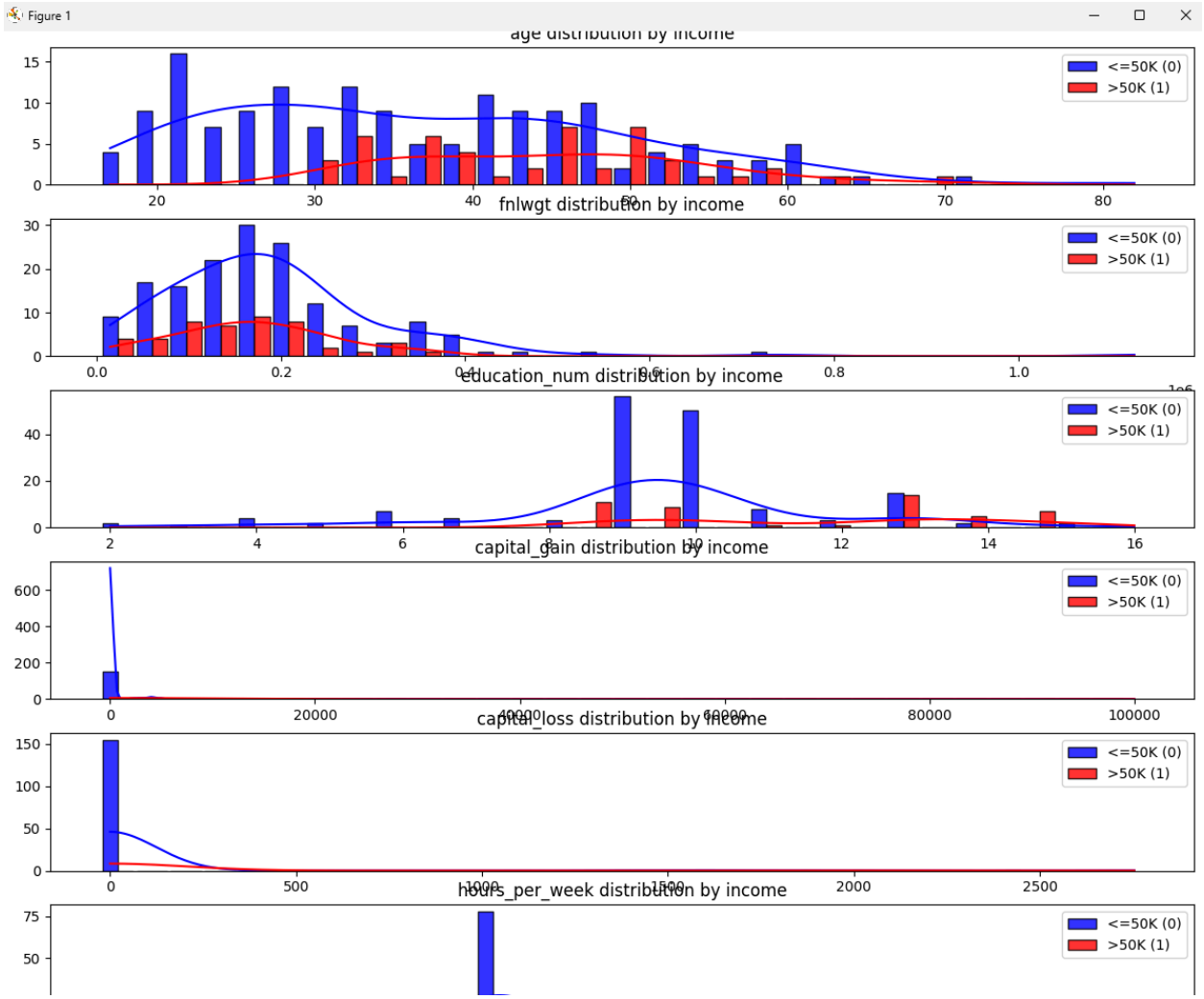
10. GitHub Link

https://github.com/ayu232/cs634_Yu_Amanda_Final_Project/tree/main

11. Appendix (Mandatory)

• Screenshots of Jupyter Notebook and .py execution (training and evaluation). • Additional plots (e.g., confusion matrices per algorithm, ROC curves). • Supporting notes or clarifications (environment details, versions, hardware).

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import gaussian_kde
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, brier_score_loss
```



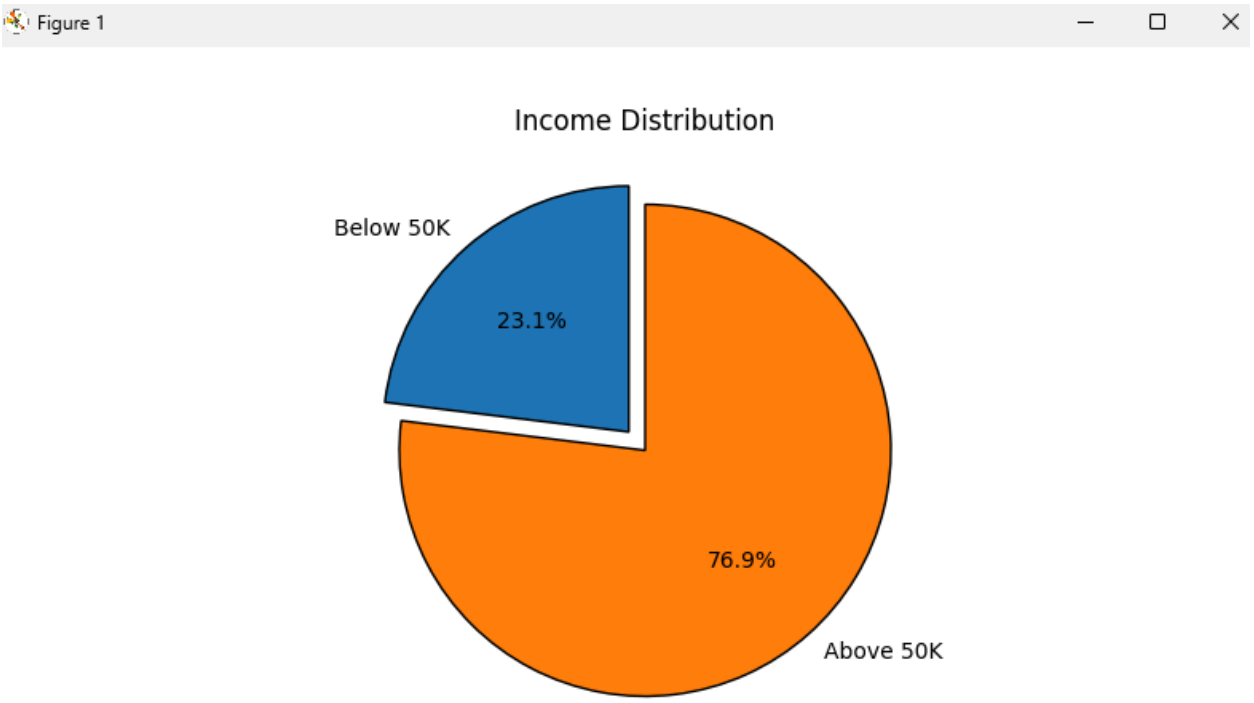
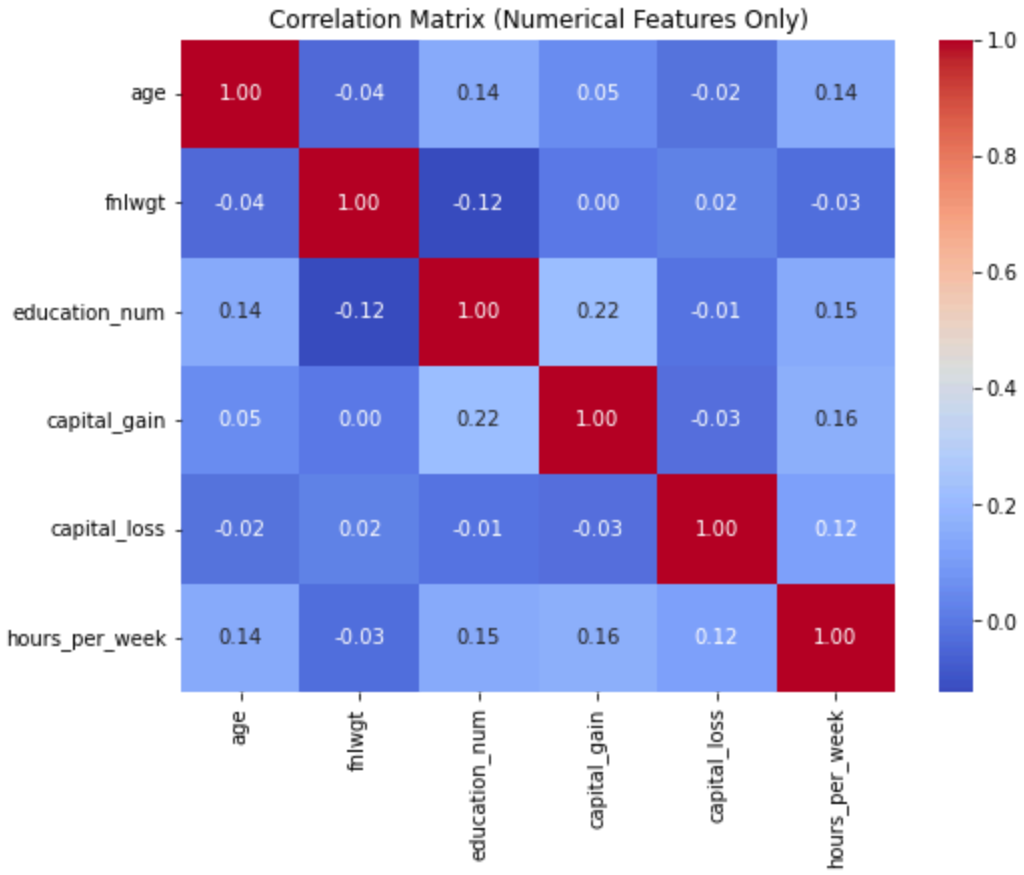
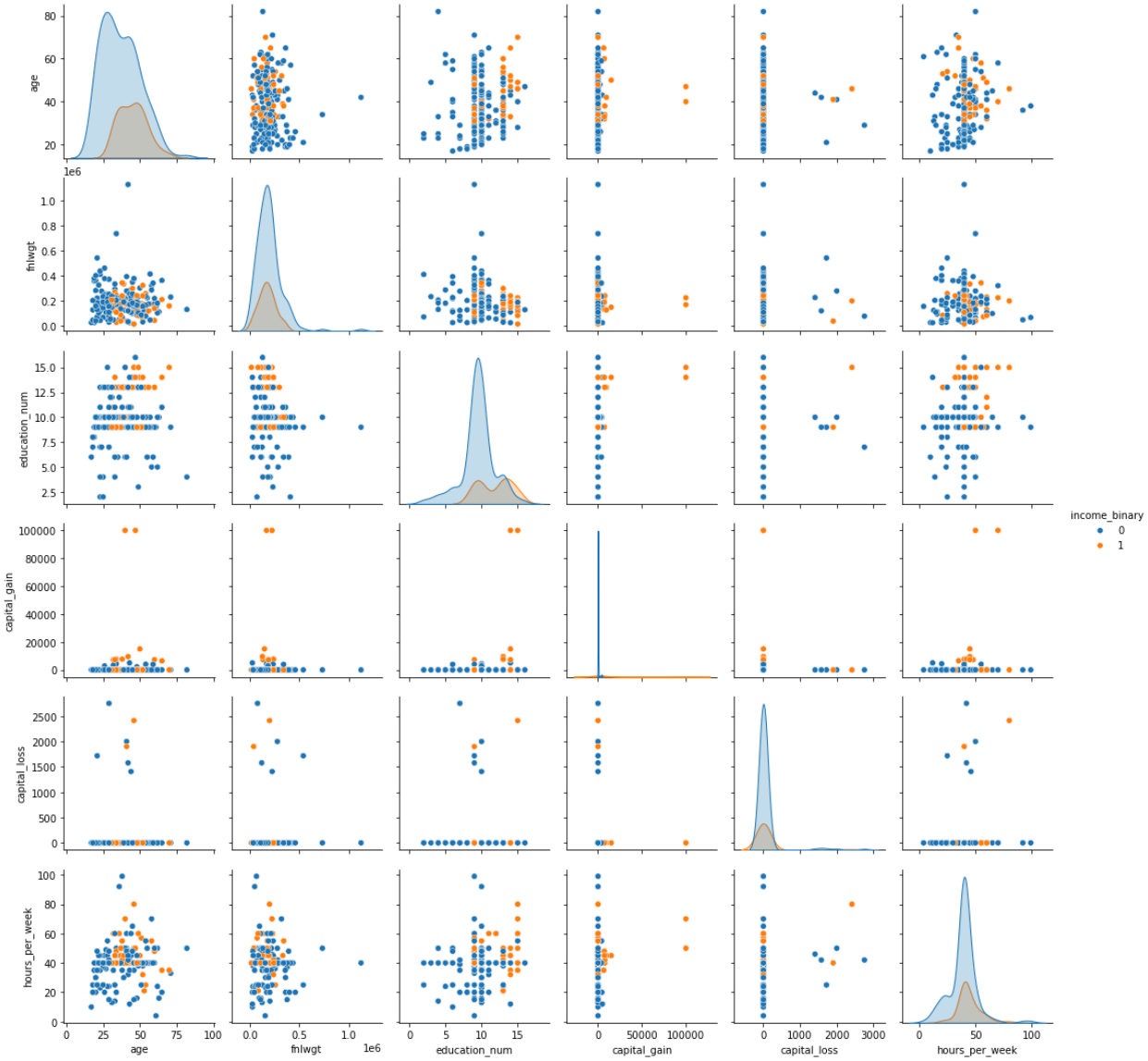
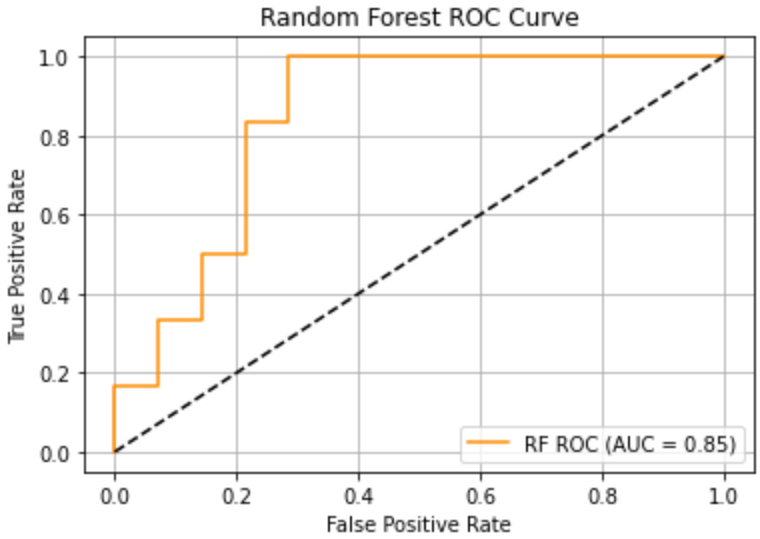
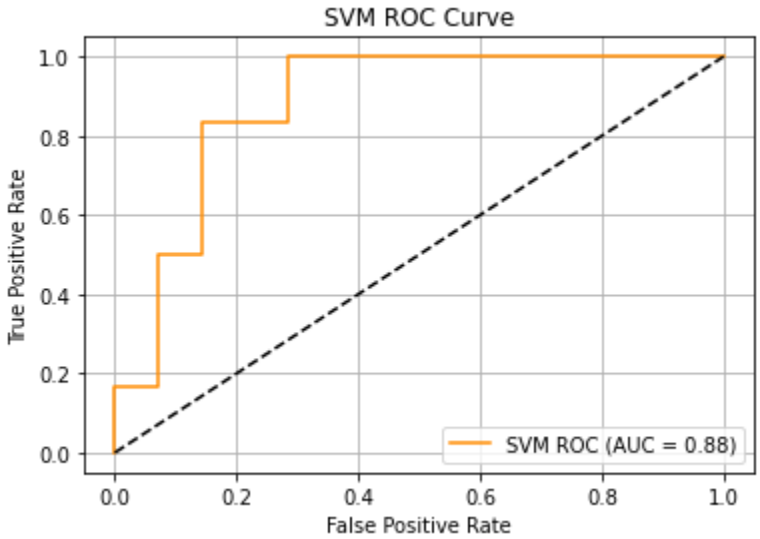


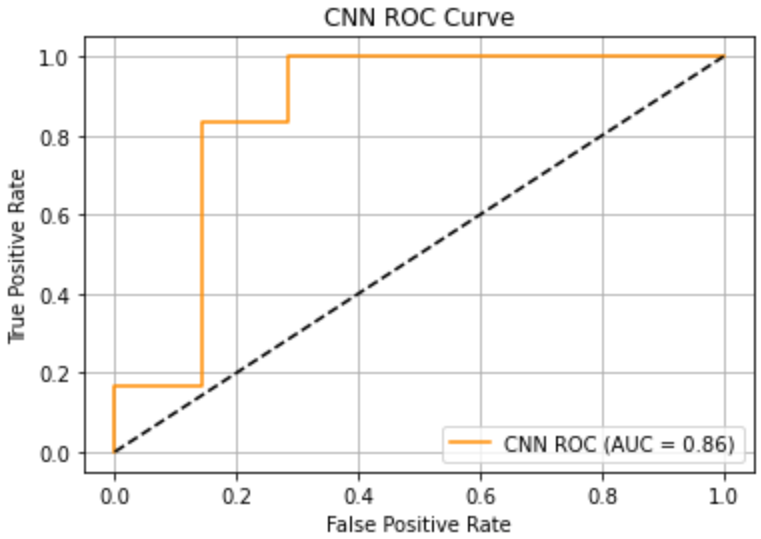
Figure 1
Income Distribution
Below 50K
23.1%
76.9%
Above 50K


Correlation Matrix (Numerical Features Only)
age
fnlwgt
education_num
capital_gain
capital_loss
hours_per_week
1.00 -0.04 0.14 0.05 -0.02 0.14
-0.04 1.00 -0.12 0.00 0.02 -0.03
0.14 -0.12 1.00 0.22 -0.01 0.15
0.05 0.00 0.22 1.00 -0.03 0.16
-0.02 0.02 -0.01 -0.03 1.00 0.12
0.14 -0.03 0.15 0.16 0.12 1.00
1.0
0.8
0.6
0.4
0.2
0.0


age
fnlwgt
1e6
education_num
capital_gain
capital_loss
hours_per_week
income_binary
0
1

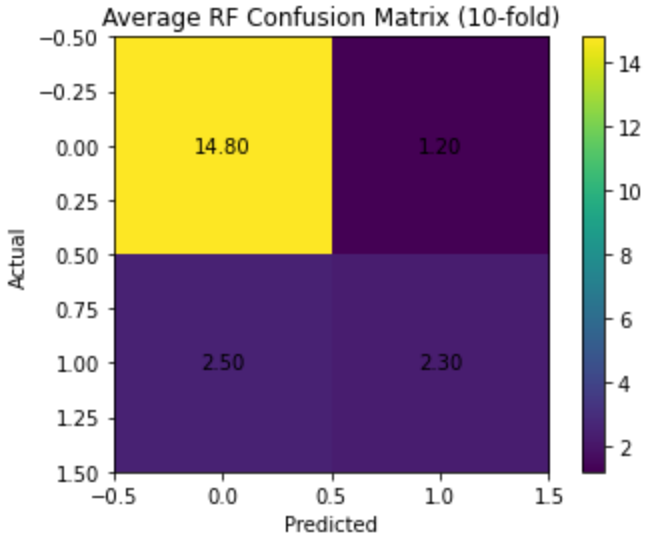

Random Forest ROC Curve
True Positive Rate
False Positive Rate
RF ROC (AUC = 0.85)
0.0
0.2
0.4
0.6
0.8
1.0

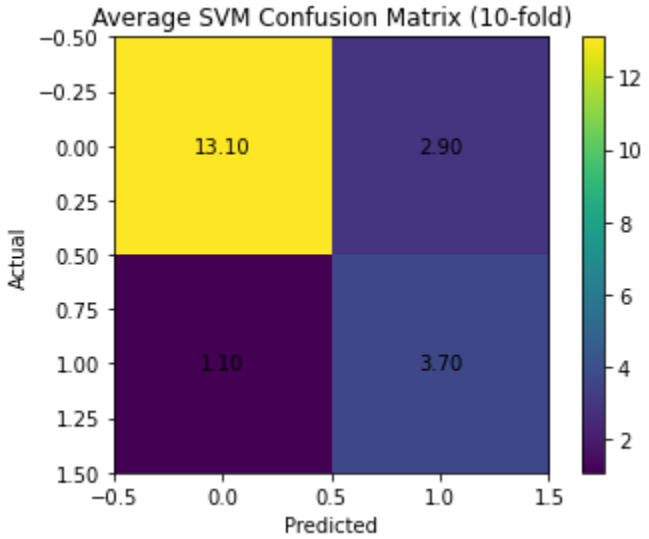

SVM ROC Curve
True Positive Rate
False Positive Rate
SVM ROC (AUC = 0.88)
0.0
0.2
0.4
0.6
0.8
1.0

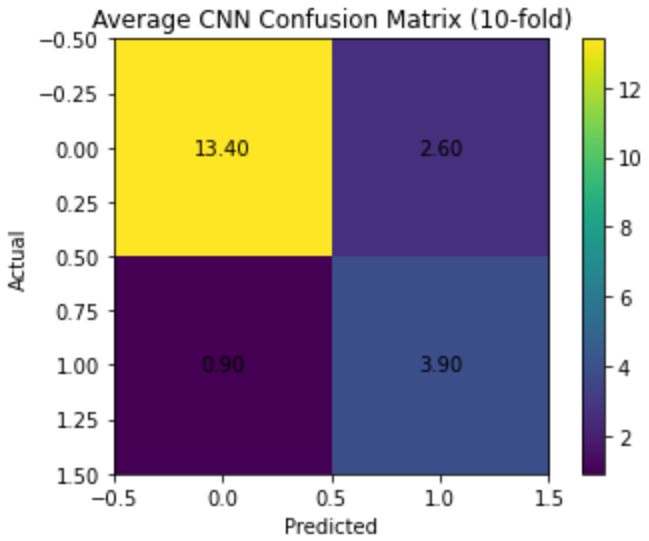


***** AVG METRICS ACROSS 10 FOLDS *****

	RF_avg	SVM_avg	CNN_avg
TP	2.30	3.70	3.90
TN	14.80	13.10	13.40
FP	1.20	2.90	2.60
FN	2.50	1.10	0.90
TPR	0.54	0.78	0.84
TNR	0.93	0.83	0.84
FPR	0.07	0.17	0.16
FNR	0.46	0.22	0.16
Precision	0.71	0.59	0.62
Accuracy	0.82	0.81	0.83
Recall	0.54	0.78	0.84
F1_measure	0.54	0.62	0.67
Error_rate	0.18	0.19	0.17
BAcc	0.73	0.80	0.84
TSS	0.47	0.60	0.68
HSS	0.44	0.51	0.57
AUC	0.91	0.90	0.93
Brier_score	0.12	0.12	0.12
BSS	0.30	0.32	0.25


Average RF Confusion Matrix (10-fold)
14.80
1.20
2.50
2.30
Actual
Predicted
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
−0.5
0.0
0.5
1.0
1.5
14
12
10
8
6
4
2


Average SVM Confusion Matrix (10-fold)
13.10
2.90
1.10
3.70
Actual
Predicted
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
−0.5
0.0
0.5
1.0
1.5
12
10
8
6
4
2


Average CNN Confusion Matrix (10-fold)
13.40
2.60
0.90
3.90
Actual
Predicted
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
−0.5
0.0
0.5
1.0
1.5
12
10
8
6
4
2