Machine Learning Analysis on AVC and Salary Datasets

Dragomir Andrei

May 26, 2024

Abstract

This document provides a comprehensive analysis of two machine learning models applied to the given datasets. The analysis includes data preprocessing, model training, evaluation, and comparison of results.

Contents

1 Introduction

In this analysis, we explore the Logistic Regression and the Multy Layer Perceptron models on the AVC and Salary datasets. The goal is to predict the target variable based on the features provided in the datasets.

Dataset Description

2 Medical and Lifestyle Information Dataset

This dataset contains medical values and relevant lifestyle information for 5110 individuals. The dataset is intended to be used for predicting whether a person is likely to have a cerebrovascular accident (stroke) or not. The target attribute is *cerebrovascular_accident*, which has binary values indicating the presence (1) or absence (0) of a stroke. The classification task is binary.

The attributes in the dataset are as follows:

Table 1: Attributes of the Medical and Lifestyle Information Dataset

Attribute	**Type**	**Description**
mean_blood_sugar_level	numeric	The average blood sugar level over the observation period.
cardiovascular_issues	categorical	Indicates if the subject has a history of cardiovascular issues. Possible values: 0, 1.
job_category	categorical	The domain in which the person works. Possible values: child, entrepreneurial, N_work_history, private_sector, public_sector.
body_mass_indicator	numeric	The body mass index indicating if the person is underweight, normal weight, overweight, or obese.
sex	categorical	The gender of the person. Possible values: F, M.
tobacco_usage	categorical	Indicator for tobacco use, present or past. Possible values: ex-smoker, smoker, non-smoker.
high_blood_pressure	categorical	Binary attribute indicating if a person suffers from high blood pressure. Possible values: 0, 1.
married	categorical	Binary attribute indicating if the person has ever been married. Possible values: Y, N.
living_area	categorical	Type of area where the person has lived most of their life. Possible values: City, Countryside.
years_old	numeric	The age of the person in years.
chaotic_sleep	categorical	Binary attribute for an irregular sleep schedule. Possible values: 0, 1.
analysis_results	numeric	Results of the person's medical tests, which may include various measurements and health indicators.
biological_age_index	numeric	An index estimating the biological age of a person based on various factors such as lifestyle and health status.
cerebrovascular_accident	categorical	Binary indicator of whether the person has had a stroke. Possible values: 0, 1.

3 Employee Information Dataset

This dataset contains personal, educational, and professional information of various employees. The objective of this dataset is the binary classification of employees into categories of earning above or below $50K per year. The classification task is binary.

The attributes in the dataset are as follows:

Table 2: Attributes of the Employee Information Dataset

Attribute	**Type**	**Description**
fnl	numeric	Socio-economic characteristic of the population from which the individual comes.
hpw	numeric	Number of hours worked per week.
relation	categorical	The type of relationship in which the individual is involved.
gain	numeric	Capital gain.
country	categorical	Country of origin.
job	categorical	The individual's occupation.
edu_int	numeric	Number of years of education.
years	numeric	Age of the individual.
loss	numeric	Capital loss.
work_type	categorical	Type of occupation.
partner	categorical	Type of partner the individual has.
edu	categorical	Type of education of the individual.
gender	categorical	Gender of the individual.
race	categorical	Race of the individual.
prod	numeric	Capital production.
gtype	categorical	Type of work contract.

4 Attribute Type Identification

Before utilizing a machine learning model for a dataset, it is crucial to identify the types of features in the dataset and their values. Understanding the nature of the attributes helps in selecting appropriate preprocessing techniques and models. The key distinctions among the types of attributes in the provided datasets are as follows:

4.1 Continuous Numerical Attributes

Continuous numerical attributes are features that can take any value within a given range. These attributes are measured on a continuous scale and can be divided into finer increments. Examples from the datasets include:

- **mean_blood_sugar_level**: Average blood sugar level measured over the observation period.
- **body_mass_indicator**: Body mass index indicating whether a person is underweight, normal weight, overweight, or obese.
- **years_old**: Age of the person in years.

- **analysis_results**: Results of medical tests.
- **biological_age_index**: An index estimating the biological age of a person based on various factors.
- **fnl**: Socio-economic characteristic of the population the individual comes from.
- **hpw**: Number of hours worked per week.
- **gain**: Capital gain.
- **edu_int**: Number of years of education.
- **years**: Age of the individual.
- **loss**: Capital loss.
- **prod**: Capital production.

4.2 Discrete Attributes

Discrete attributes are features that take on a finite number of distinct values. These values are often categorical and can be counted in whole numbers. Examples from the datasets include:

- **cardiovascular_issues**: Whether the subject has a history of cardiovascular issues (0 or 1).
- **sex**: Gender of the person (F or M).
- **tobacco_usage**: Indicator for smokers, either past or present (ex-smoker, smoker, non-smoker).
- **high_blood_pressure**: Indicator if a person has high blood pressure (0 or 1).
- **married**: Whether the person has ever been married (Y or N).
- **living_area**: Type of area where the person has lived most of their life (City or Countryside).
- **chaotic_sleep**: Indicator for an irregular sleep schedule (0 or 1).
- **cerebrovascular_accident**: Indicator if the person has had a stroke (0 or 1).
- **relation**: Type of relationship the individual is involved in.
- **country**: Country of origin.
- **job**: Job of the individual.
- **work_type**: Type of job.
- **partner**: Type of partner the individual has.
- **edu**: Type of education of the individual.

- **gender**: Gender of the individual.
- **race**: Race of the individual.
- **gtype**: Type of work contract.
- **job_category**: Domain in which the person works.

Recognizing these distinctions helps in choosing the right methods for handling the data during preprocessing. For instance, continuous numerical attributes might require normalization or standardization, discrete attributes might need encoding, and ordinal attributes might need ordinal encoding to maintain the order information. Properly identifying and classifying the attributes ensures that the machine learning models can effectively learn from the data and make accurate predictions.

5 Numeric Attributes Analysis

In this section, we analyze the numeric attributes of the two datasets provided. The analysis includes the number of non-missing values, mean value, standard deviation, minimum value, 25th percentile, 50th percentile (median), 75th percentile, and maximum value for each numeric attribute.

5.1 Healthcare Dataset

Table 3: Statistics of Numeric Attributes in the Healthcare Dataset

Attribute	**No-miss**	**Mean**	**Std Dev**	**Min**	**25th Pctl**	**Mid**	**75th Pctl**	**Max**
mean_blood_sugar_level	5110	106.15	45.28	55.12	77.25	91.89	114.09	271.74
body_mass_indicator	4909	28.89	7.85	10.30	23.50	28.10	33.10	97.60
years_old	5110	46.57	26.59	0.08	26.00	47.00	63.75	134.00
analysis_results	4599	323.52	101.58	104.83	254.65	301.03	362.82	756.81
biological_age_index	5110	134.78	50.40	-15.11	96.71	136.37	172.51	266.99

Comments:

- The **mean_blood_sugar_level** has a mean of 106.15 with a standard deviation of 45.28, indicating a wide range of values.
- The **body_mass_indicator** shows a mean of 28.89, suggesting that on average, individuals fall into the overweight category.
- The **years_old** attribute ranges from 0.08 to 134 years, with a median of 47 years.
- The **analysis_results** attribute has significant variability, as indicated by its standard deviation of 101.58.
- The **biological_age_index** has negative values, which may need to be investigated further for data correctness.

5.2 Employee Dataset

Table 4: Statistics of Numeric Attributes in the Employee Dataset

Attribute	No-miss	Mean	Std Dev	Min	25th Pctl	Mid	75th Pctl	Max
fnl	9999	190352.9	106070.8	19214	118282.5	178472	237311	1455435
hpw	9199	40.42	12.52	1.0	40.0	40.0	45.0	99.0
gain	9999	979.85	7003.80	0.0	0.0	0.0	0.0	99999.0
edu_int	9999	14.26	24.77	1.0	9.0	10.0	13.0	206.0
years	9999	38.65	13.75	17.0	28.0	37.0	48.0	90.0
loss	9999	84.11	394.04	0.0	0.0	0.0	0.0	3770.0
prod	9999	2014.93	14007.60	-28.0	42.0	57.0	77.0	200125.0

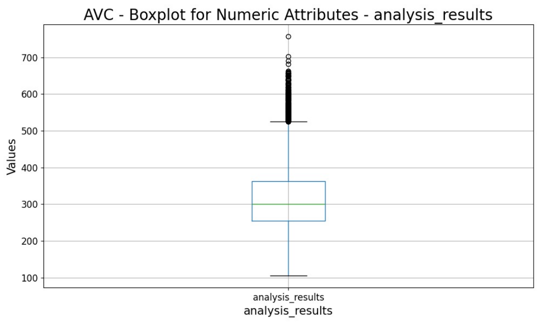
Comments:

- The **fnl** attribute has a high mean value and a wide range, indicating significant differences in the socio-economic status of individuals.
- The **hpw** (hours per week) attribute shows that most individuals work around 40 hours per week.
- The **gain** and **loss** attributes have a high standard deviation, indicating that only a few individuals have large capital gains or losses.
- The **edu_int** (years of education) attribute shows that the majority of individuals have around 10 to 13 years of education.
- The **years** attribute shows an average age of around 38.65 years, with a minimum of 17 and a maximum of 90 years.
- The **prod** attribute has a high standard deviation, indicating variability in capital production.

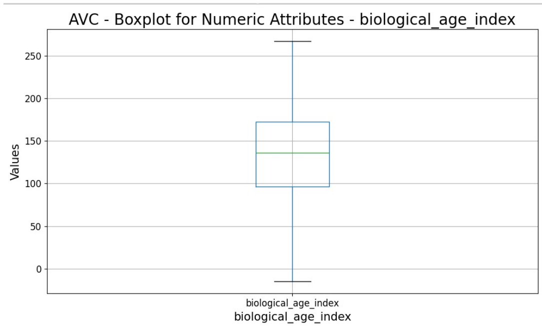
6 Numeric Ranges for Attributes

In this section, we present the boxplots for the numeric attributes in the datasets to visualize the distribution and identify any potential outliers.

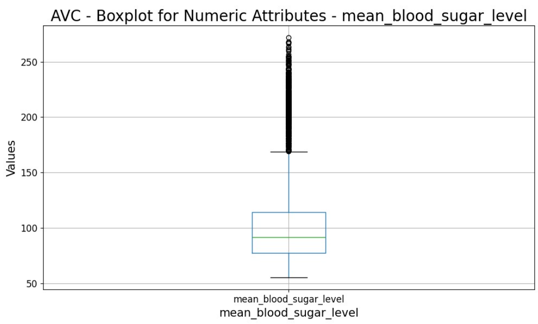
6.1 Healthcare Dataset



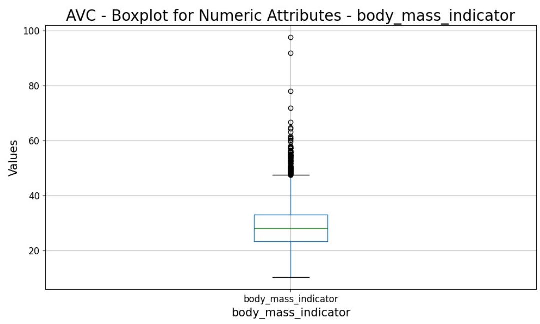
(a) Analysis Results



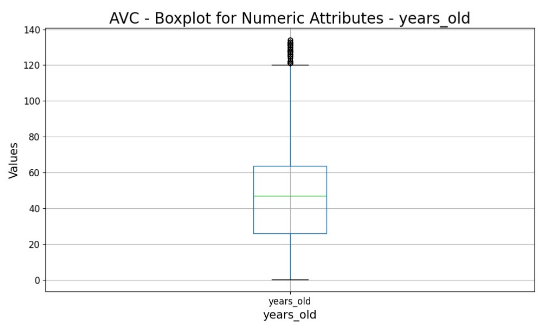
(b) Biological Age Index



(c) Mean Blood Sugar Level



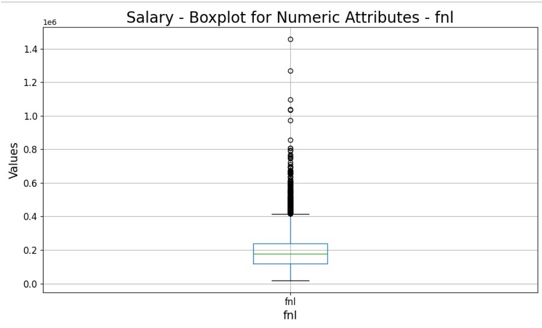
(d) Body Mass Indicator



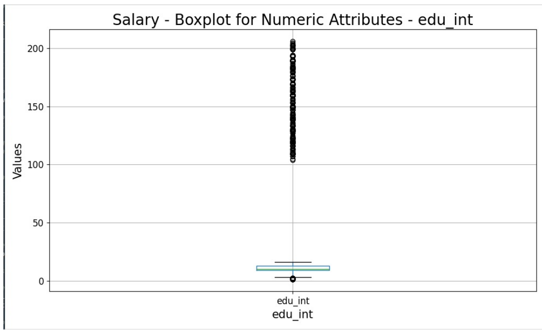
(e) Years Old

Figure 1: Boxplots for Healthcare Dataset Numeric Attributes

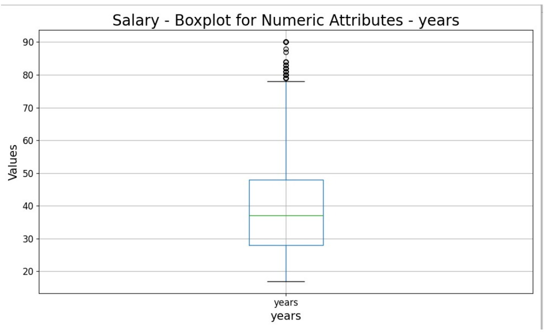
6.2 Employee Dataset



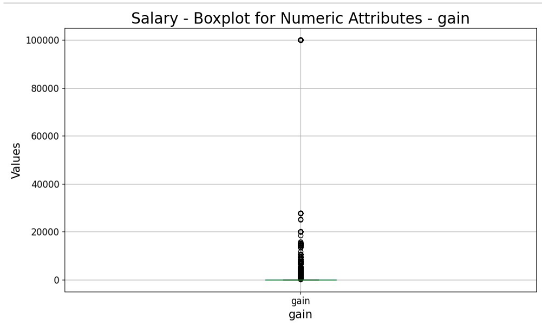
(a) FNL



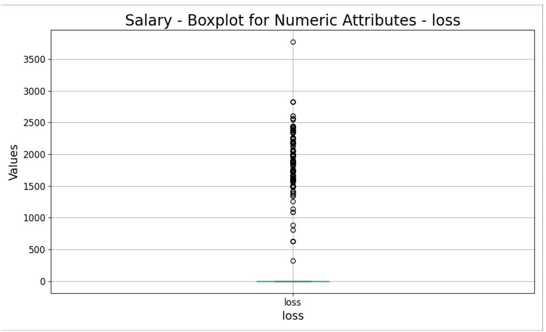
(b) Education (Years)



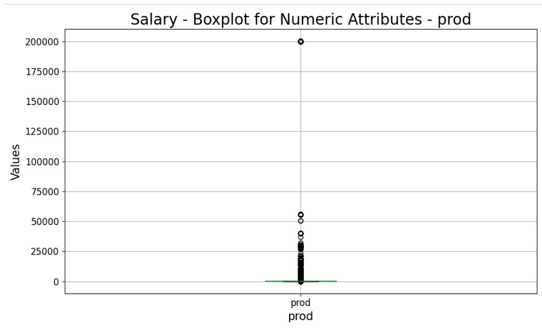
(c) Years



(d) Gain



(e) Loss



(f) Prod

Figure 2: Boxplots for Employee Dataset Numeric Attributes

7 Categorical Attributes Analysis

In this section, we analyze the categorical attributes of the datasets. The tables below present the attributes along with the number of non-missing values and the number of unique values for each attribute.

7.1 Healthcare Dataset

Table 5: Categorical Attributes - Healthcare Dataset

Attribute	**Number of Non-Missing Values**	**Number of Unique Values**
cardiovascular_issues	5110	2
job_category	5110	5
sex	5110	2
tobacco_usage	5110	4
high_blood_pressure	5110	2
married	4599	2
living_area	5110	2
chaotic_sleep	5110	2
cerebrovascular_accident	5110	2

7.2 Employee Dataset

Table 6: Categorical Attributes - Employee Dataset

Attribute	**Number of Non-Missing Values**	**Number of Unique Values**
relation	9999	6
country	9999	41
job	9999	14
work_type	9999	9
partner	9999	7
edu	9999	16
gender	9199	2
race	9999	5
gtype	9999	2
money	9999	2

7.3 Comments

The healthcare dataset includes several categorical attributes such as ‘cardiovascular_issues‘, ‘job_category‘, ‘sex‘, and ‘tobacco_usage‘. Each of these attributes has a varying number of unique values, indicating different levels of categorization. For instance, ‘tobacco_usage‘ has four unique values, while ‘sex‘ has only two. It is also noted that the ‘married‘ attribute has fewer non-missing values compared to others, which might require special handling during data preprocessing.

The employee dataset, on the other hand, contains a wider range of categorical attributes with a larger variety of unique values, especially for ‘country‘ and ‘edu‘, which have 41 and 16 unique values, respectively. Attributes like ‘gender‘ and ‘gtype‘ are binary, which simplifies their processing. The variety in the number of unique values across different attributes in both datasets indicates the complexity and diversity of the data, which must be carefully considered during the feature engineering and model building stages.

8 Categorical Distribution for Attributes

8.1 Healthcare Dataset

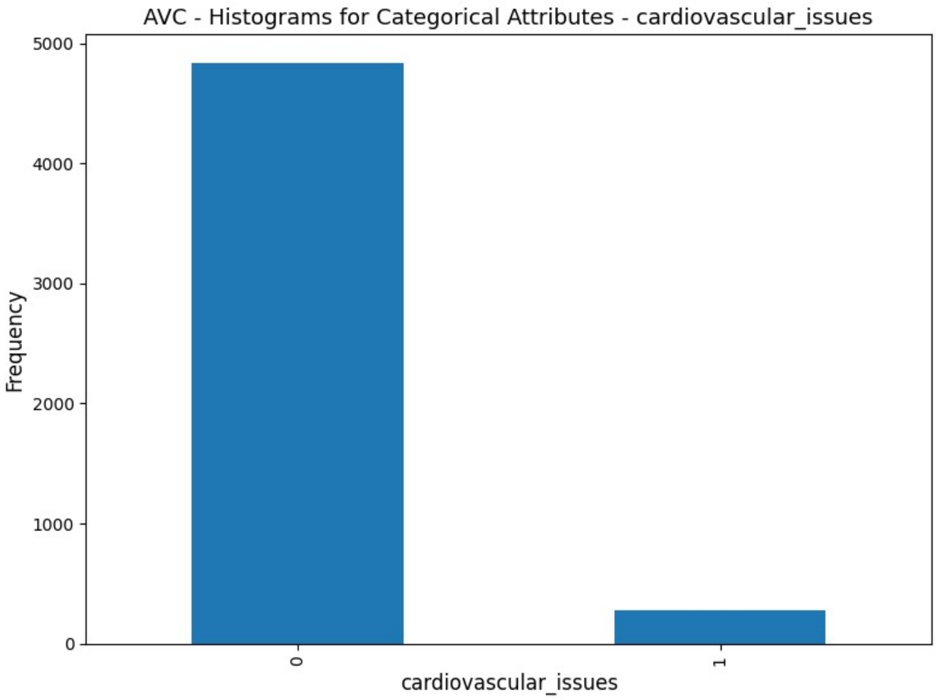


Figure 3: Cardiovascular Issues

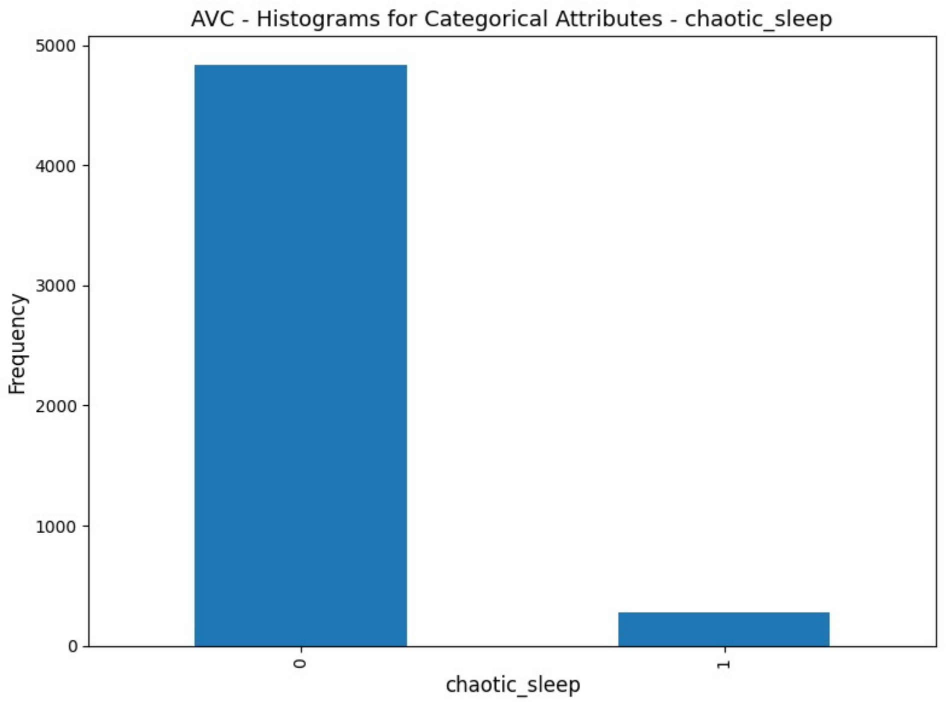


Figure 4: Chaotic Sleep

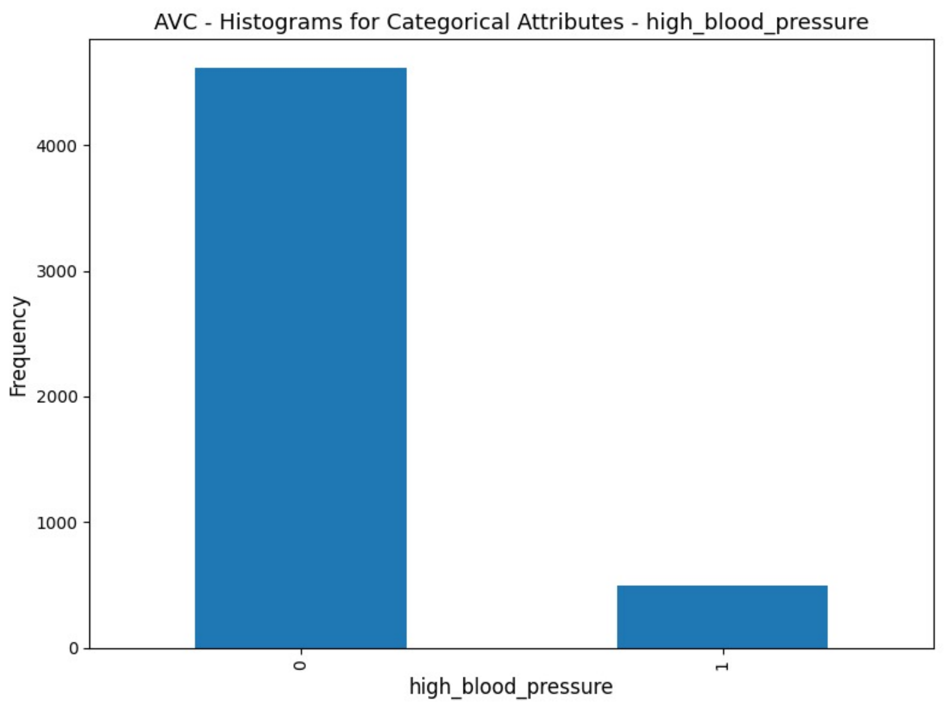


Figure 5: High Blood Pressure

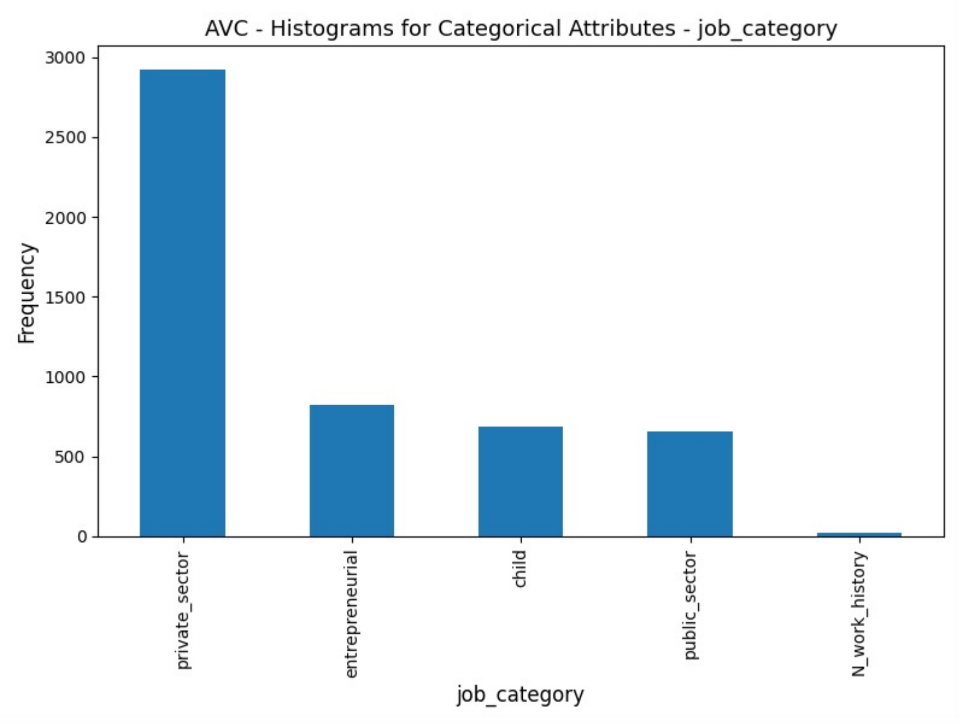


Figure 6: Job Category

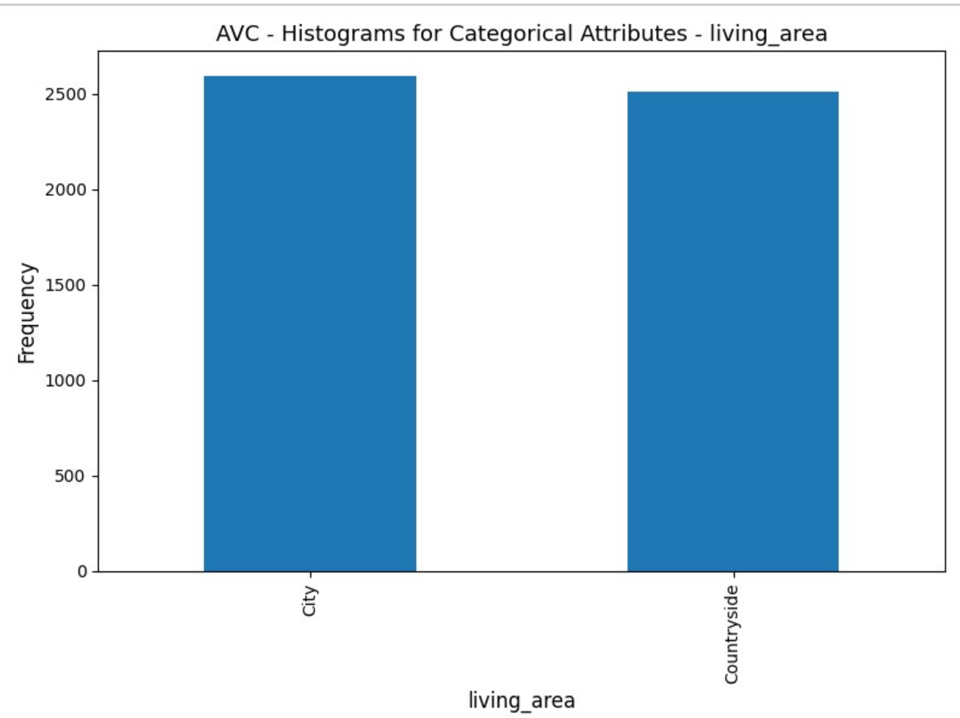


Figure 7: Living Area

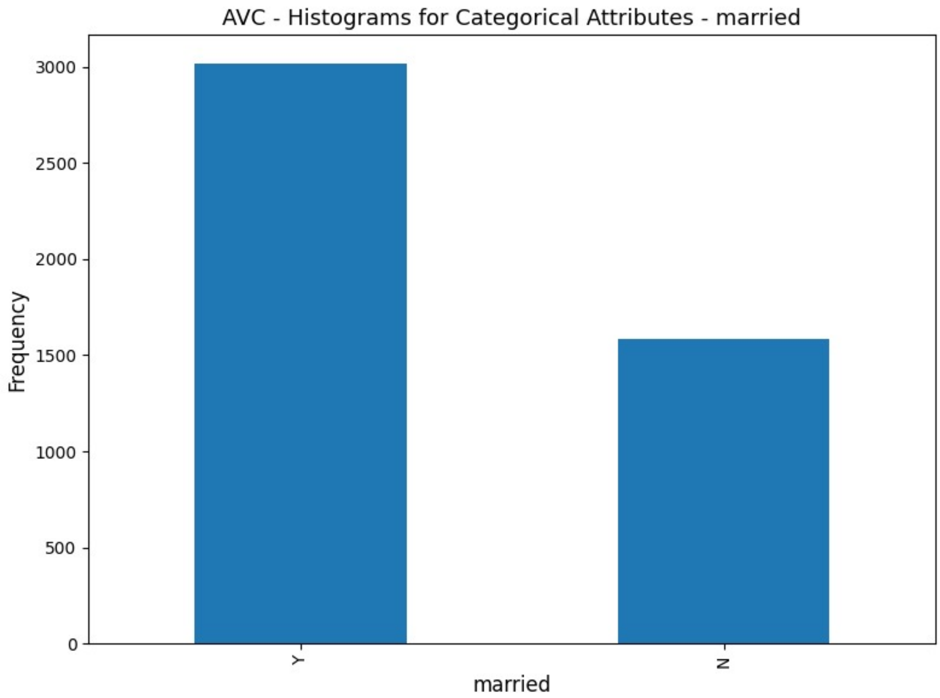


Figure 8: Married

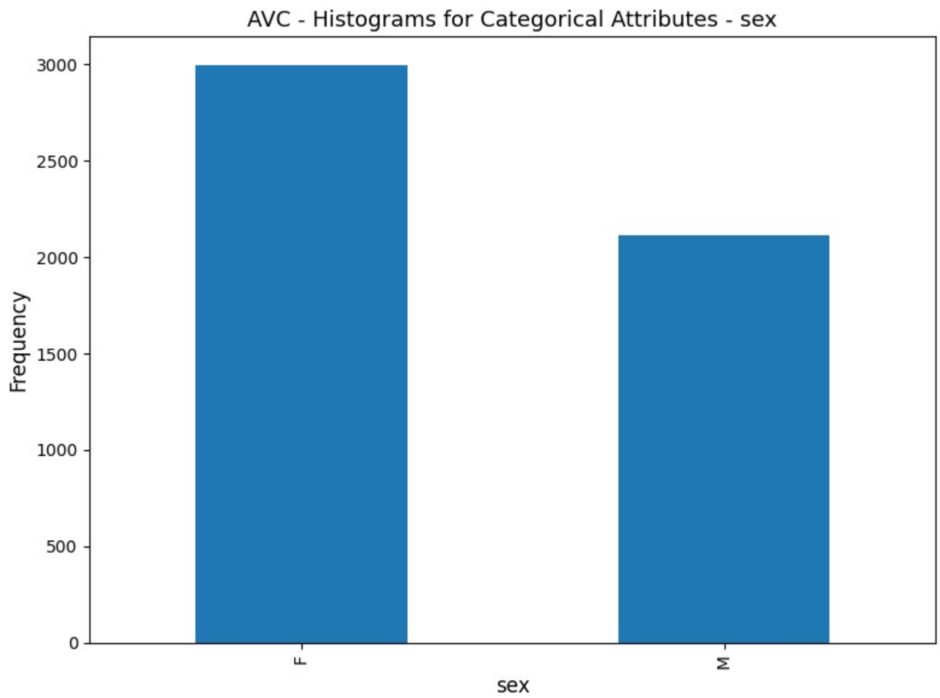


Figure 9: Sex

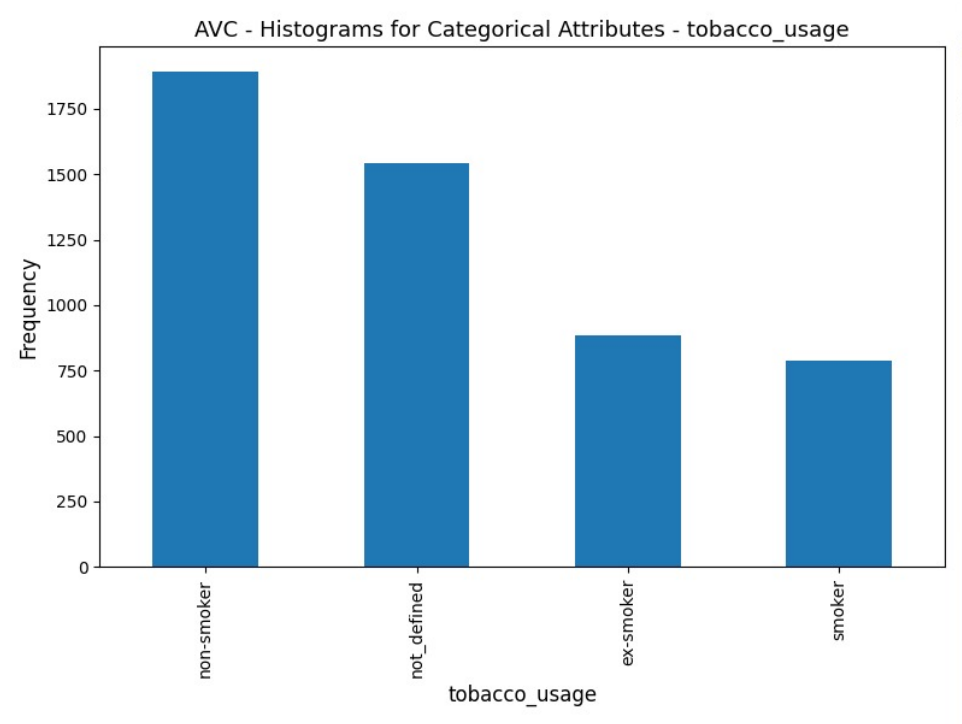


Figure 10: Tobacco Usage

8.2 Salary Dataset

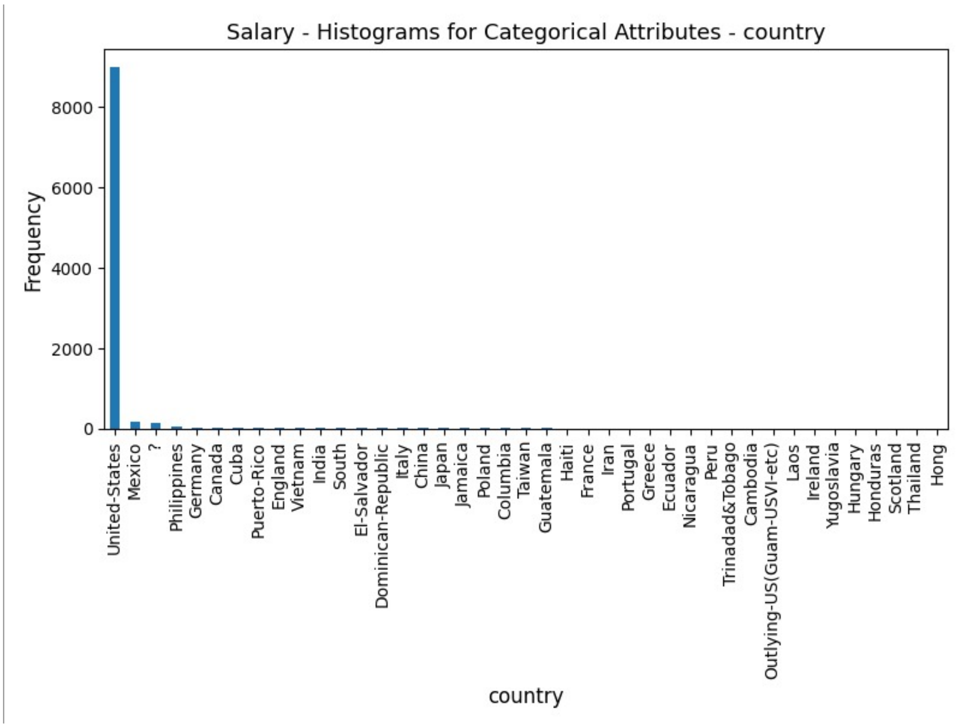


Figure 11: Country

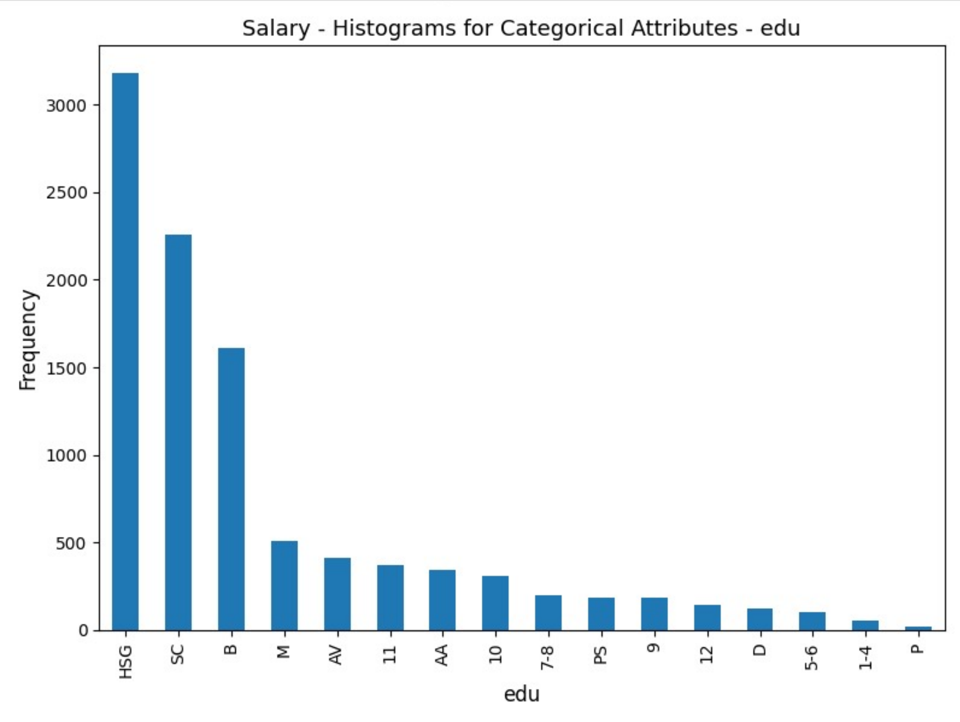


Figure 12: Education

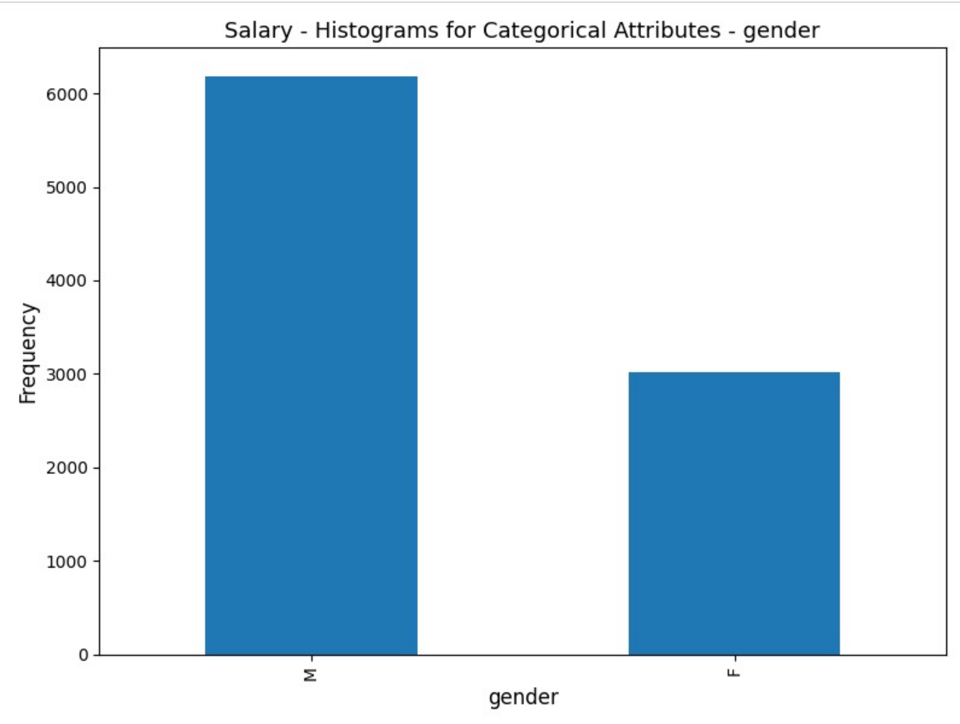


Figure 13: Gender

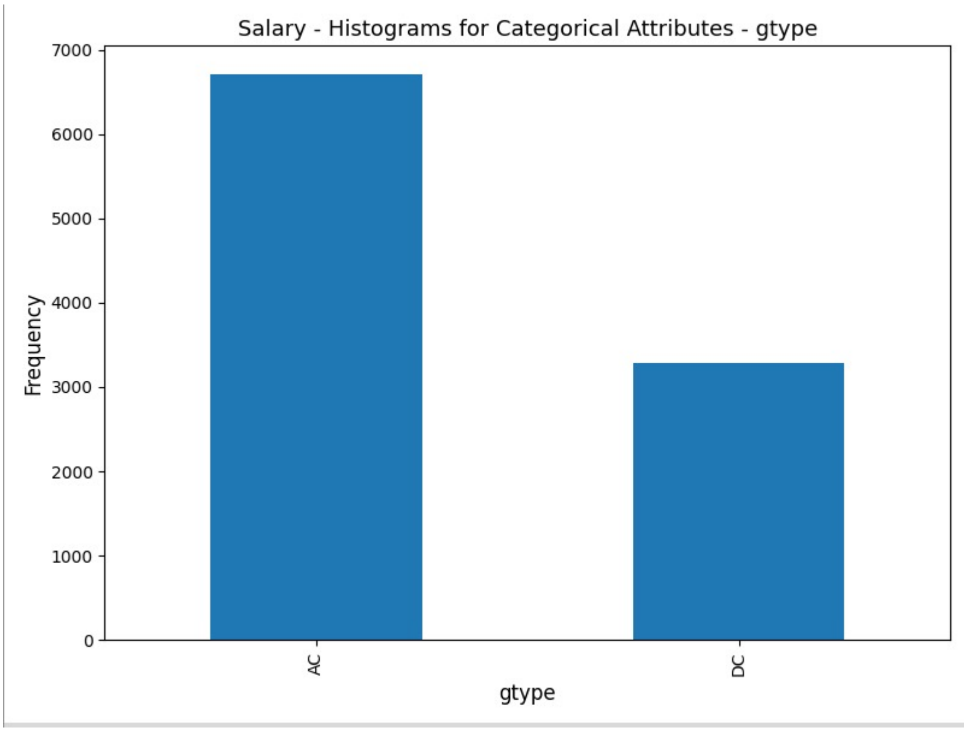


Figure 14: Job Type

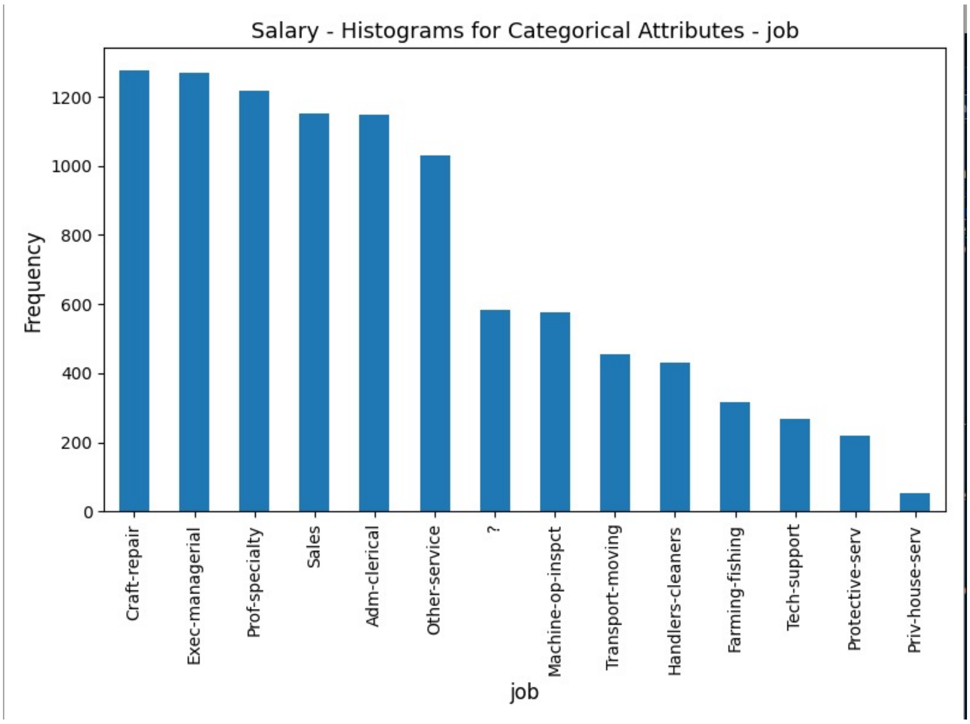


Figure 15: Job

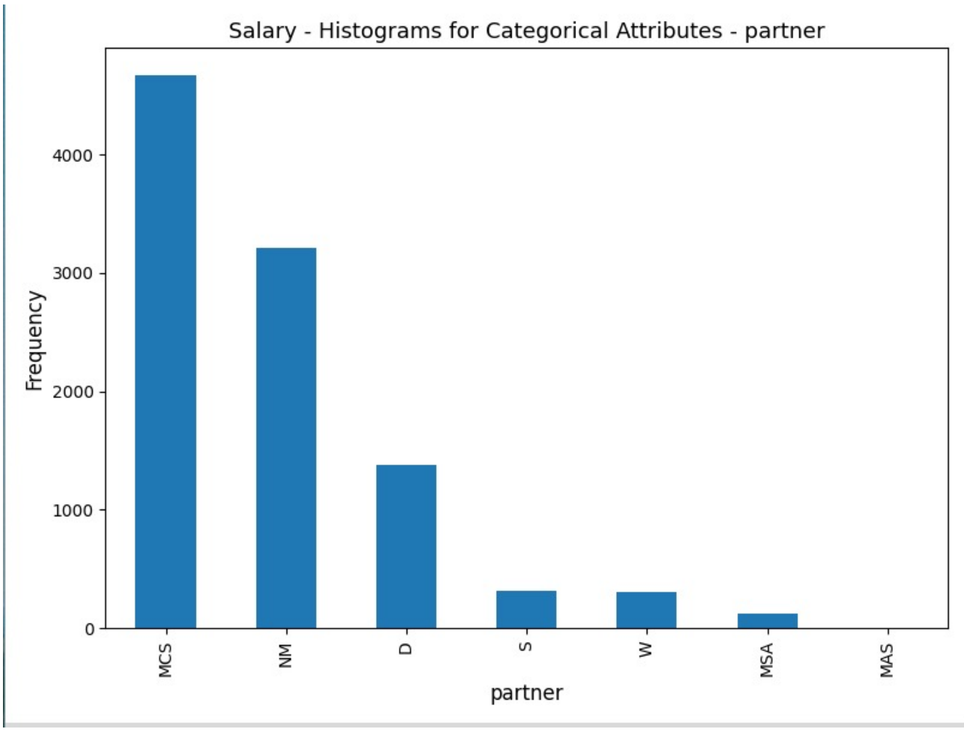


Figure 16: Partner

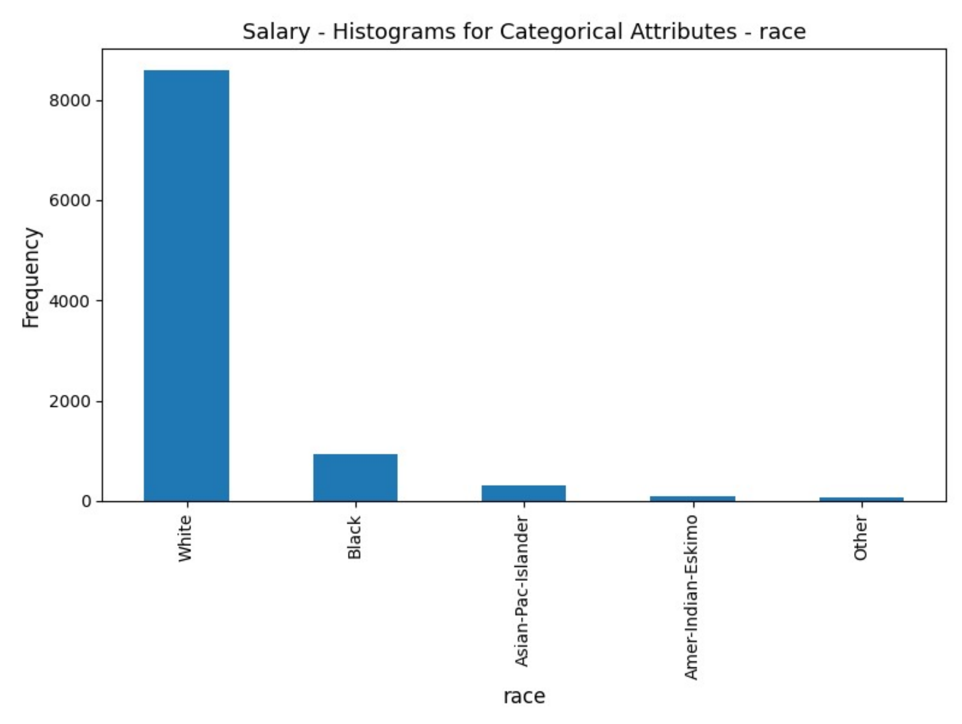


Figure 17: Race

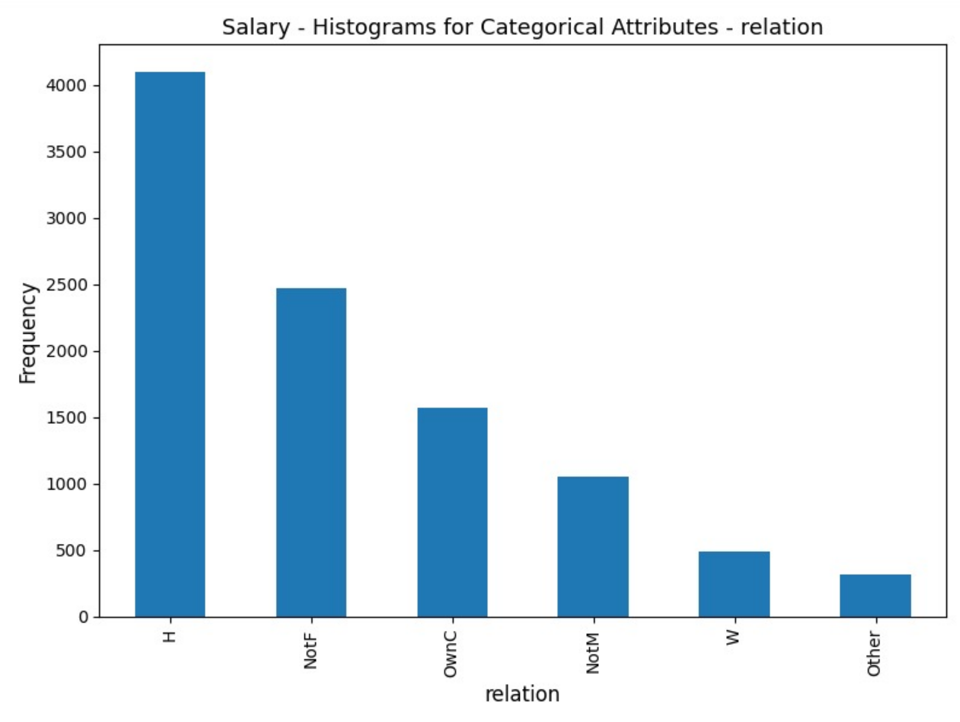


Figure 18: Relation

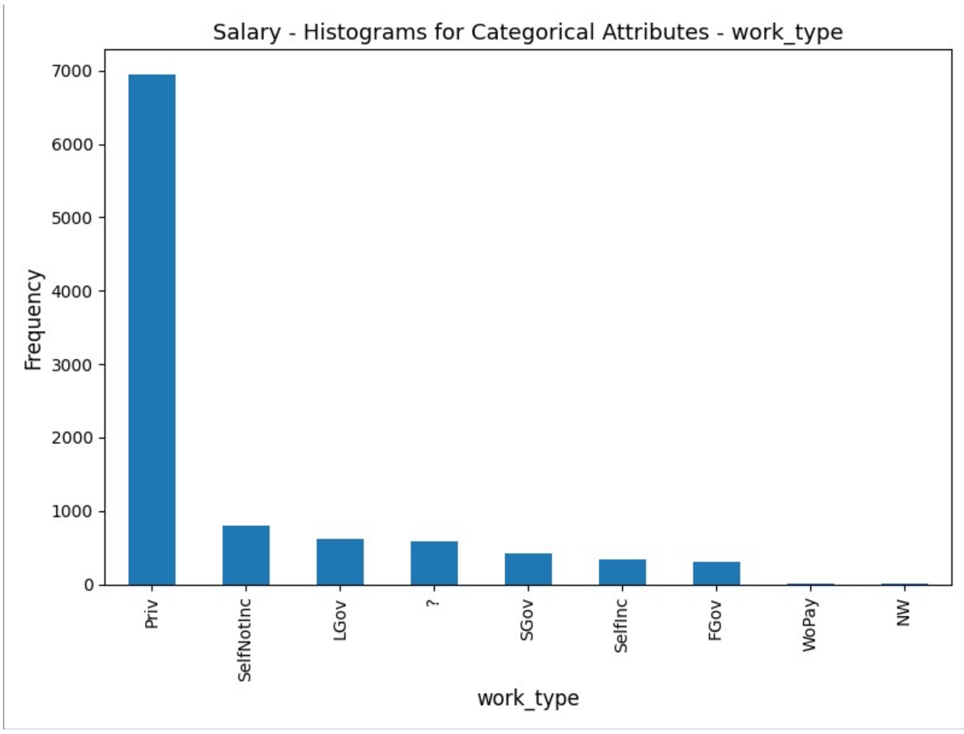


Figure 19: Work Type

9 Class Balance

This section presents the class balance for both the AVC and Salary datasets. The class balance is an important aspect to consider as it influences the evaluation metrics we should focus on.

9.1 AVC Dataset

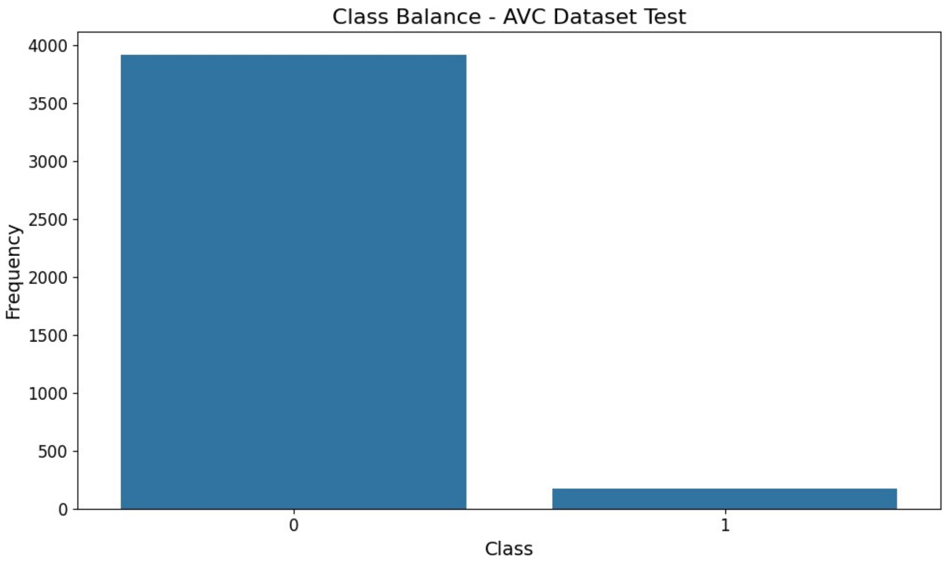


Figure 20: Class Balance - AVC Dataset Test

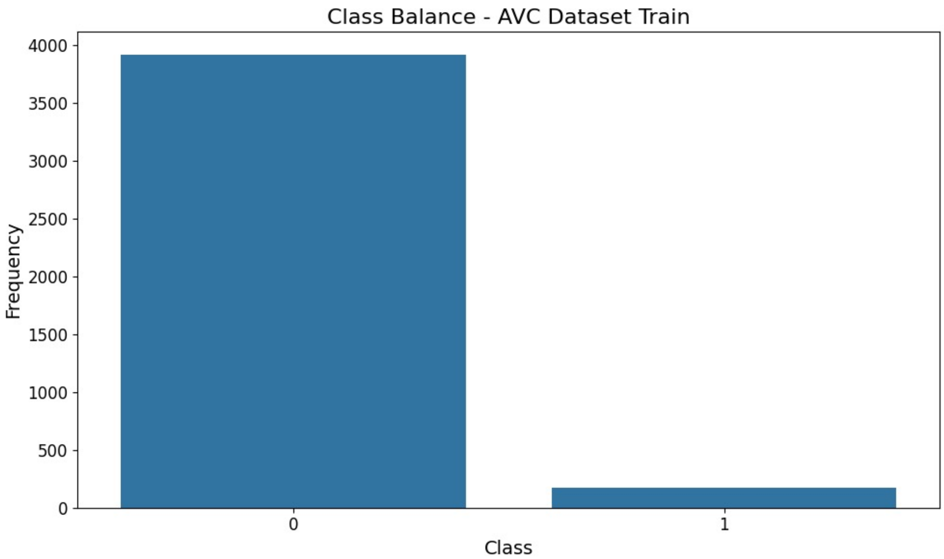


Figure 21: Class Balance - AVC Dataset Train

9.2 Salary Dataset

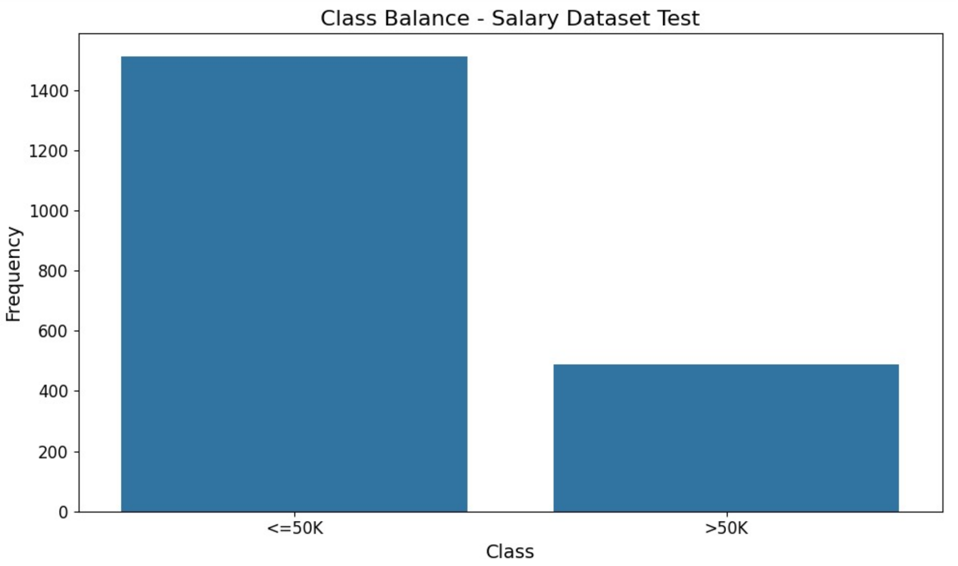


Figure 22: Class Balance - Salary Dataset Test

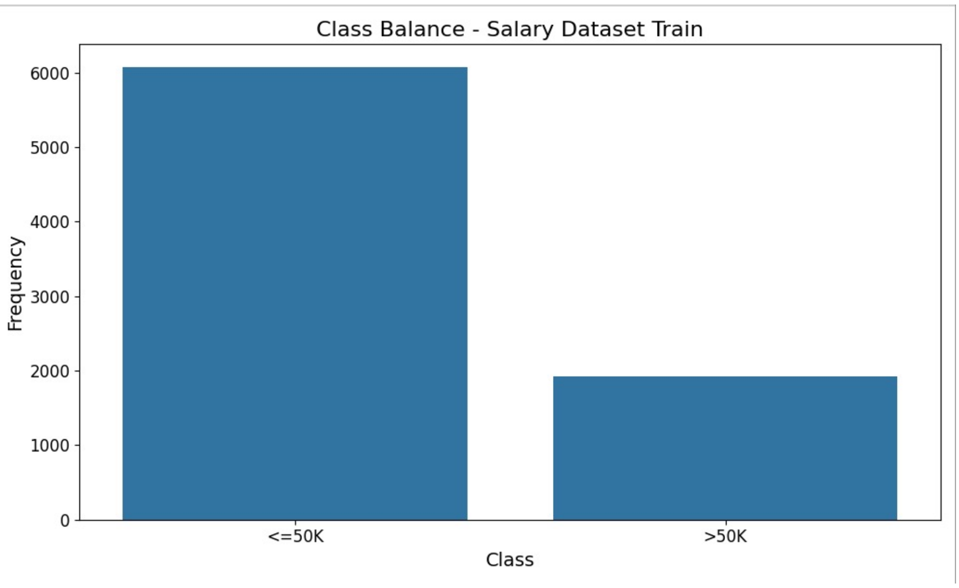


Figure 23: Class Balance - Salary Dataset Train

9.3 Comments

From the class balance plots above, it is evident that both datasets exhibit significant class imbalance. In the AVC dataset, the majority class (no stroke) is much more prevalent than the minority class (stroke). Similarly, in the Salary dataset, the class of individuals earning less than or equal to 50K is more common than the class of individuals earning more than 50K.

Class imbalance can have a substantial impact on model performance and evaluation. Models may become biased towards the majority class, leading to poor performance on the minority class. To address this, it is crucial to focus on evaluation metrics that provide a better understanding of performance on imbalanced datasets. Specifically, we should emphasize:

- **F1 Score**: The harmonic mean of precision and recall, which provides a balance between the two.
- **Precision**: The ability of the classifier to not label a negative sample as positive.
- **Recall**: The ability of the classifier to find all positive samples.

By prioritizing these metrics, we can better assess and compare the performance of different algorithms on these imbalanced datasets.

10 Attribute Correlation

This section presents the correlation matrices for both the AVC and Salary datasets. The correlation matrices are split into numerical and categorical attributes for both train and test sets.

10.1 AVC Dataset

10.1.1 Numerical Attributes

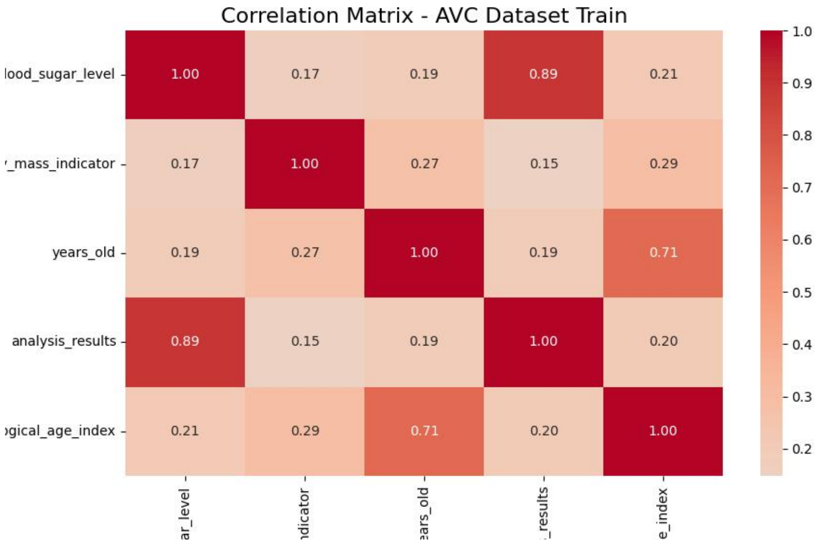


Figure 24: Correlation Matrix - Numerical Attributes - AVC Dataset Train

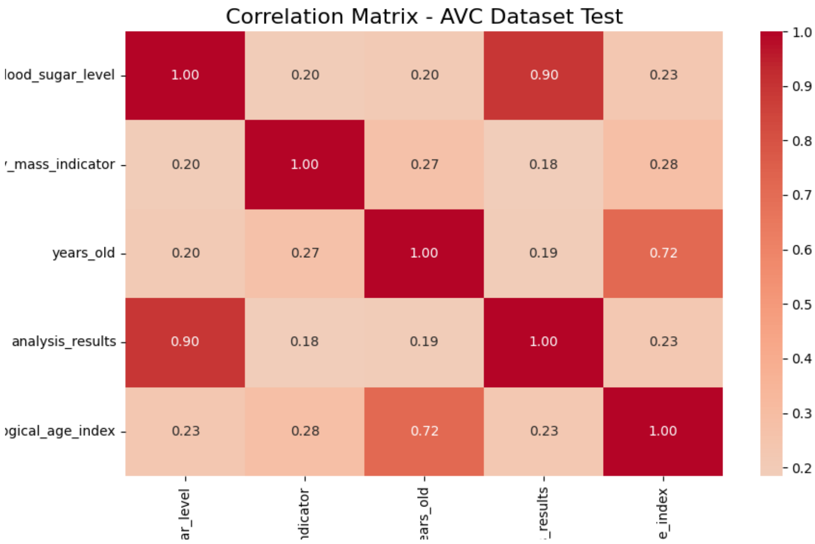


Figure 25: Correlation Matrix - Numerical Attributes - AVC Dataset Test

10.1.2 Categorical Attributes

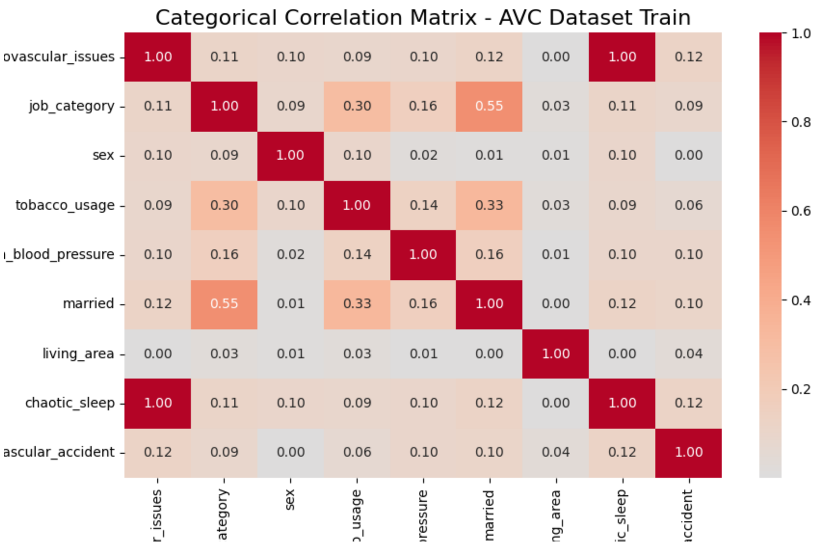


Figure 26: Categorical Correlation Matrix - AVC Dataset Train

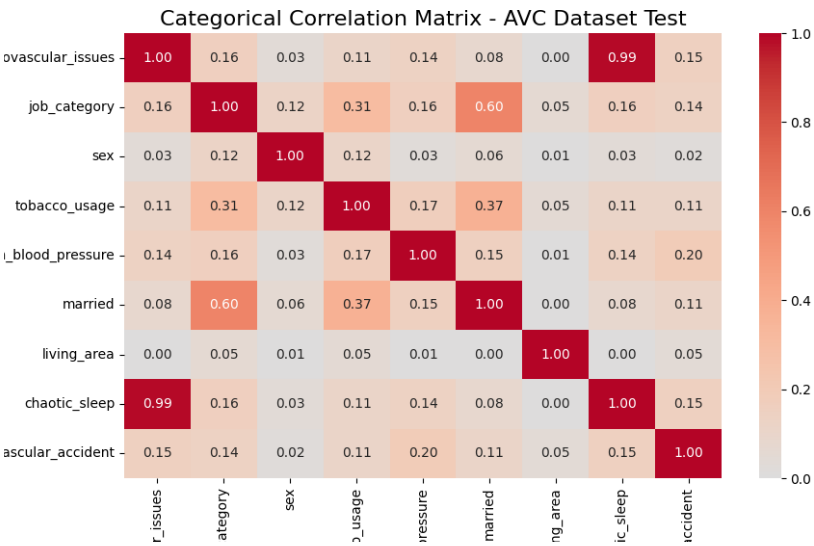


Figure 27: Categorical Correlation Matrix - AVC Dataset Test

10.2 Salary Dataset

10.2.1 Numerical Attributes

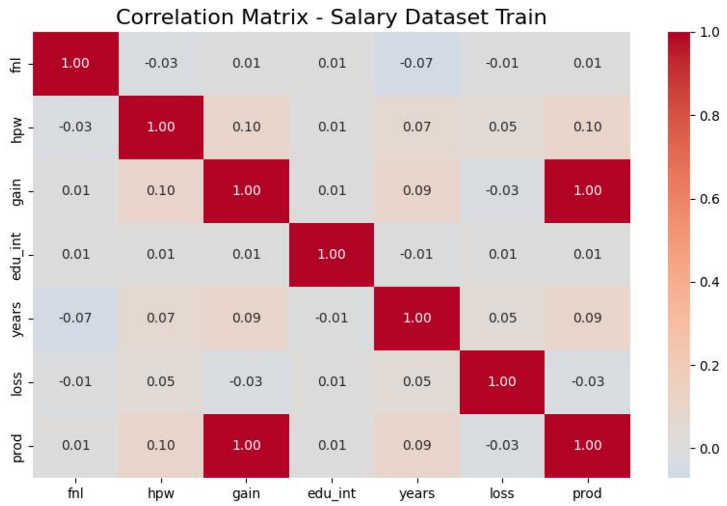


Figure 28: Correlation Matrix - Numerical Attributes - Salary Dataset Train

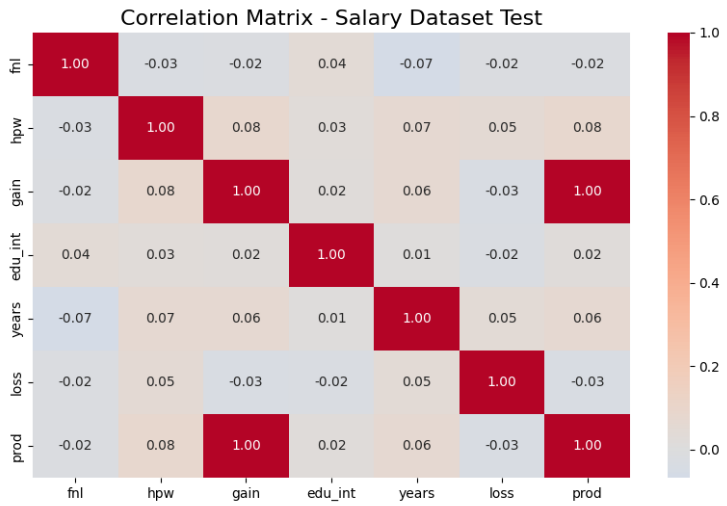


Figure 29: Correlation Matrix - Numerical Attributes - Salary Dataset Test

10.2.2 Categorical Attributes

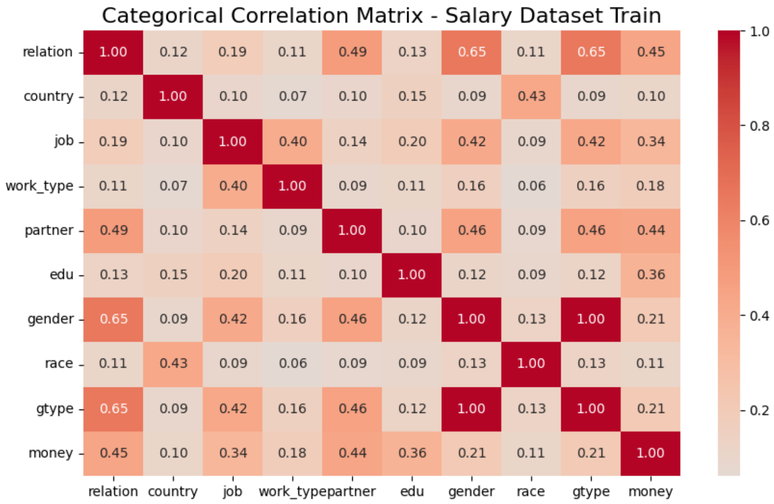


Figure 30: Categorical Correlation Matrix - Salary Dataset Train

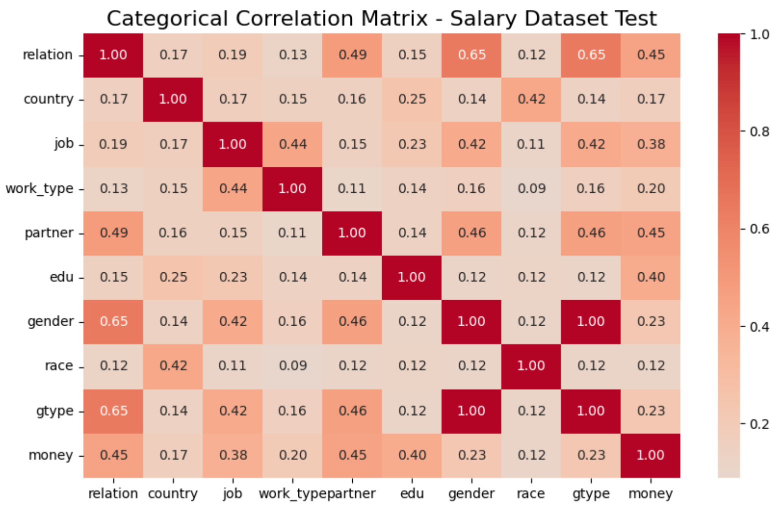


Figure 31: Categorical Correlation Matrix - Salary Dataset Test

10.3 Comments

The correlation matrices provide insights into the relationships between different attributes in the datasets.

10.3.1 AVC Dataset

Numerical Attributes From the numerical attribute correlation matrices of the AVC dataset, it is observed that:

- There is a moderate positive correlation between *years_old* and *biological_age_index* in both train and test sets, which indicates that as the age of the person increases, the biological age index also tends to be higher.
- *Analysis_results* show a high correlation with *blood_sugar_level*, but low with other numerical attributes, suggesting that the results of medical analyses are relatively independent of the person's age and other indicators.

Categorical Attributes The categorical correlation matrices for the AVC dataset indicate:

- There are some moderate correlations between categorical variables such as *married* and *job_category*, and *job_category* and *tobacco_usage*.
- These correlations suggest that there might be certain patterns or dependencies between the demographic and lifestyle attributes.
- There is a high correlation between *chaotic_sleep* and *high_blood_pressure*, indicating that individuals with irregular sleep schedules are more likely to have high blood pressure.

10.3.2 Salary Dataset

Numerical Attributes For the Salary dataset, the numerical attribute correlation matrices reveal:

- A strong positive correlation between *gain* and *prod*, indicating that individuals who gain more also tend to have higher produce of capital.
- *Hours per week (hpw)* shows a low correlation with most other numerical attributes, suggesting that the number of hours worked per week is not strongly related to other numerical factors like capital gains or losses.

Categorical Attributes In the categorical correlation matrices for the Salary dataset:

- Moderate correlations are observed between *work_type* and *job*, as well as between *country* and *race*.
- These correlations highlight possible demographic and professional patterns.
- There is also a high correlation between *gtype* and *gender*, indicating that the type of work contract is directly influenced by the gender.

Overall, these correlation matrices help in understanding the interdependencies between attributes, which can be useful in feature selection and in improving the performance of machine learning models.

Eliminating specific variables such as *analysis_results* and *biological_age_index* from the AVC dataset, and prod from the Salary dataset, can enhance the efficiency and performance of our machine learning models. These variables were identified as candidates for elimination due to their high correlation with other attributes, which makes them redundant, and their minimal impact on predictive accuracy. By removing these redundant or less impactful features, we reduce noise, prevent overfitting, and simplify the model, leading to improved generalization and more robust predictive performance. This streamlined approach allows the model to focus on the most significant attributes, enhancing both computational efficiency and model interpretability.

11 Preprocess of Data

Preprocessing data is a crucial step in machine learning. It involves cleaning, transforming, and encoding data to make it suitable for modeling. Below are the steps and functions used in our preprocessing pipeline.

11.1 Imputing Missing Values

Missing values are imputed using the mean for numeric columns and the most frequent value for categorical columns.

```
def impute_missing_values(df, nmr_cols, cat_cols, fit, imp):
    df_nmr = df[nmr_cols]
    df_cat = df[cat_cols]
    df.replace('?', np.nan, inplace=True)
    df.replace('not_defined', np.nan, inplace=True)
    if fit:
        # Fit imputer if on training data
        imp = SimpleImputer(strategy='mean')
        df_nmr_imputed = pd.DataFrame(imp.fit_transform(df_nmr), cols=nmr_cols)
        imp = SimpleImputer(strategy='most_frequent')
        df_cat_imputed = pd.DataFrame(imp.fit_transform(df_cat), cols=cat_cols)
    else:
        # Use the fitted imputer from training data
        df_nmr_imputed = pd.DataFrame(imp['nmr'].transform(df_nmr), cols=nmr_cols)
        df_cat_imputed = pd.DataFrame(imp['cat'].transform(df_cat), cols=cat_cols)

    df_imputed = pd.concat([df_nmr_imputed, df_cat_imputed], axis=1)
    return df_imputed, imputers
```

11.2 Handling Extreme Values

Extreme values are handled using the Interquartile Range (IQR) method and then imputed.

```
def handle_extreme_values(df, nmr_cols):
    # Handling extreme values using IQR method
    Q1 = df[nmr_cols].quantile(0.25)
    Q3 = df[nmr_cols].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Replace extreme values with NaN
    for col in nmr_cols:
        df[col] = np.where((df[col] < lower_bound[col]) |
                            (df[col] > upper_bound[col]),
                            np.nan, df[col])

    # Impute missing values for the added NaNs
    imputer = SimpleImputer(strategy='mean')
    df[nmr_cols] = imputer.fit_transform(df[nmr_cols])
    return df
```

11.3 Removing Highly Correlated Attributes

Attributes with high or medium high correlation (above 0.5) are removed to prevent multicollinearity. Correlation classes:
-0 to 0.3: low correlation
-0.3 to 0.7: medium correlation
-0.7 to 1: high correlation

This resulted in the following eliminations: **analysis_results** and **biological_age_index** from the AVC dataset, and **prod** from the Salary dataset.

```
def remove_redundant_attributes(df, nmr_cols, threshold=0.9):
    numeric_df = df[nmr_cols]

    # Get upper triangle of correlation matrix
    corr_mat = numeric_df.corr().abs()
    upper = corr_mat.where(np.triu(np.ones(corr_mat.shape), k=1).astype(bool))

    # Drop everything higher than threshold
    to_drop = [col for col in upper.cols if any(upper[col] > threshold)]
    df_reduced = df.drop(cols=to_drop)
    return df_reduced, to_drop
```

11.4 Standardizing Numerical Attributes

Standardizing ensures that the numerical attributes have a mean of 0 and a standard deviation of 1.

```
def standardize_data(df, nmr_cols, method='standard', fit=True, scaler=None):
    if fit:
        if method == 'standard':
            scaler = StandardScaler()
        elif method == 'minmax':
            scaler = MinMaxScaler()
        elif method == 'robust':
            scaler = RobustScaler()
        df[nmr_cols] = scaler.fit_transform(df[nmr_cols])
        return df, scaler
    else:
        df[nmr_cols] = scaler.transform(df[nmr_cols])
        return df
```

11.5 Encoding Categorical Variables

Categorical variables are encoded using OneHotEncoder, and the target variable is encoded using LabelEncoder.

```
def encode_categorical(df, trg_col, enc=None):
    df = df.copy()
    cat_cols = df.select_dtypes(include=['object']).cols.tolist()

    # Remove target column from categorical columns
    if trg_col in categorical_cols:
        cat_cols.remove(trg_col)

    # Encode target column
    label_enc = LabelEncoder()
    df[trg_col] = label_enc.fit_transform(df[trg_col])

    # Encode categorical columns
    enc = OneHotEncoder(sparse_output=False, drop='first', handle_unknown='ignore')
    df_onehot = pd.DataFrame(enc.fit_transform(df[categorical_cols]),
                             cols=enc.get_feature_names_out(categorical_cols))

    df = pd.concat([df, df_onehot], axis=1)
    df.drop(categorical_cols, axis=1, inplace=True)

    return df, label_enc, enc
```

11.6 Wrapper Function for Preprocessing

The wrapper function applies the above steps to each dataset (training and test) and saves the processed data.

```
def preprocess_data_wrapper():
    for name, df in datasets.items():
        if 'train' in name:
            df_reduced, dropped_cols = rm_redundant(..., threshold=0.5)
            nmr_cols_after_drop = [col for col in nmr_cols if col not in dropped]

            df_imputed, imputer = inpute_missing(...)
            fitted_imputers[name] = imputer

            df_outliers_handled = handle_extreme_values(...)

            df_standardized, scaler = standardize_data(...)
            fitted_scalers[name] = scaler

            df_encoded, label_enc, onehot_enc = encode_categorical(...)
            fitted_encs[name] = (label_enc, onehot_enc)

            processed_datasets[name] = df_encoded
        else:
            dropped = [col for col in nmr_cols if col not in proc_df.cols]
            nmr_cols_after_drop = [col for col in nmr_cols if col not in dropped]

            imputer = fitted_imputers[train_name]
            scaler = fitted_scalers[train_name]

            df_imputed, _ = impute_missing_values(...)
            df_outliers_handled = handle_extreme_values(...)
            df_standardized = standardize_data(...)
            df_encoded, _, _ = encode_categorical(...)

            train_cols = processed_datasets[train_name].cols
            for col in train_cols:
                if col not in df_encoded.cols:
                    df_encoded[col] = 0
            df_encoded = df_encoded[train_cols]

            processed_datasets[name] = df_encoded

    X_avc_train, T_avc_train, _, _ = preprocess_data(...)
    X_avc_test, T_avc_test, _, _ = preprocess_data(...)
    X_salary_train, T_salary_train, _, _ = preprocess_data(...)
    X_salary_test, T_salary_test, _, _ = preprocess_data(...)
    return return_tuple_avc, return_tuple_salary
```

```
def preprocess_data(df, target_column, enc=None):
    # Preprocess the data by encoding cat variables and separating features
    df_enc, label_enc, enc = encode_categorical(df, target_column, enc)
    # Separate features
    X = df_enc.drop(columns=[target_column]).values
    # Separate target
    T = df_enc[target_column].values
    return X, T, label_enc, enc
```

12 Logistic Regression

12.1 Manual Implementation

In this section, we implement logistic regression from scratch. The steps involved are as follows:

- **Logistic Function**: Apply the logistic (sigmoid) function to transform input values into probabilities.
- **Negative Log-Likelihood (NLL)**: Compute the NLL, which measures how well the predicted probabilities match the true labels.
- **Accuracy**: Calculate the accuracy of the predictions by comparing them to the true labels.
- **Prediction**: Make predictions using the logistic regression model.
- **Training and Evaluation**: Train the logistic regression model using gradient descent and evaluate its performance.

```
function logistic(x):
    # Sigmoid function
    return 1 / (1 + exp(-x))

function nll(Y, T):
    # Negative Log-Likelihood
    N = number of samples
    return -sum(T * log(Y) + (1 - T) * log(1 - Y)) / N

function accuracy(Y, T):
    # Accuracy calculation
    N = number of samples
    acc = 0

    # Compare predictions to true labels
    for i in range(N):
        if (Y[i] >= 0.5 and T[i] == 1) or (Y[i] < 0.5 and T[i] == 0):
            acc += 1
    return acc / N
```

```
function predict_logistic(X, w):
    # Make predictions using logistic regression model
    return logistic(dot(X, w))

function train_and_eval_logistic(X_train, T_train, X_test, T_test, lr, epochs_no):
    N, D = shape of X_train
    w = random weights
    train_acc, test_acc = []
    train_nll, test_nll = []

    for epoch in range(epochs_no):
        # Make predictions
        Y_train = predict_logistic(X_train, w)
        Y_test = predict_logistic(X_test, w)

        # Calculate metrics
        train_acc.append(accuracy(Y_train, T_train))
        test_acc.append(accuracy(Y_test, T_test))
        train_nll.append(nll(Y_train, T_train))
        test_nll.append(nll(Y_test, T_test))

        # Update weights using gradient descent
        w = w - lr * dot(transpose(X_train), (Y_train - T_train)) / N

    return w, train_nll, test_nll, train_acc, test_acc
```

12.2 Scikit-learn Implementation

In this section, we use the `scikit-learn` library to implement logistic regression. The steps involved are:

- **Model Initialization**: Initialize the logistic regression model with a maximum of 1000 iterations.
- **Model Fitting**: Fit the model to the training data.
- **Prediction**: Make predictions on the training and test sets.
- **Evaluation**: Calculate accuracy and NLL for the training and test sets.

```
function train_and_eval_sklearn_logistic(X_train, T_train, X_test, T_test, C, solver)
    # Initialize logistic regression model
    model = LogisticRegression(penalty=penalty, C=C, solver=solver, max_iter=1000)

    # Fit the model
    model.fit(X_train, T_train)

    # Make predictions
    Y_train = model.predict_proba(X_train)[:, 1]
    Y_test = model.predict_proba(X_test)[:, 1]
```

```
    # Calculate metrics
    train_acc = accuracy_score(T_train, (Y_train >= 0.5).astype(int))
    test_acc = accuracy_score(T_test, (Y_test >= 0.5).astype(int))
    train_nll = log_loss(T_train, Y_train)
    test_nll = log_loss(T_test, Y_test)

    return model, train_nll, test_nll, train_acc, test_acc
```

12.3 Particularities

In this section, we document the specific approach and settings used for our logistic regression algorithm.

12.3.1 Categorical Attribute Encoding

For encoding categorical attributes, we employed the following approach:

- **Label Encoding**: The target variable was encoded using Label Encoding, which converts categorical labels into numerical values.
- **One-Hot Encoding**: Other categorical attributes were encoded using One-Hot Encoding. This method creates binary columns for each category, representing the presence or absence of a specific category in the data.

12.3.2 Optimization Algorithm Settings

We utilized gradient descent as the optimization algorithm with the following settings:

- **Learning Rate**: A learning rate of 0.3 was chosen to control the step size during the update of the weights. This value was selected based on preliminary experiments to balance convergence speed and stability.
- **Epochs**: The number of epochs was set to 1000 to allow the model to converge to an optimal solution. This value was chosen based on the convergence behavior observed during training.

13 MLP

13.1 Activation Functions

The following activation functions are used in our MLP implementation:

- **Sigmoid**: $\sigma(x) = \frac{1}{1+e^{-x}}$
- **ReLU (Rectified Linear Unit)**: $\text{ReLU}(x) = \max(0, x)$
- **Tanh**: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **Leaky ReLU**: $\text{Leaky ReLU}(x) = \max(\alpha x, x)$
- **ELU (Exponential Linear Unit)**: $\text{ELU}(x) = x$ if $x > 0$, $\alpha(e^x - 1)$ if $x \leq 0$

13.2 Manual Implementation

In this section, we manually implement an MLP with the following components:

- **Activation Functions**: Sigmoid, ReLU, Tanh, Leaky ReLU, and ELU
- **Layers**: Linear layer and Dropout layer
- **Loss Function**: Cross-entropy loss
- **Optimizer**: SGD and Adam optimizers

```
class Linear:
    def __init__(input_dim, output_dim, l2_reg=0.01):
        Initialize weights and biases

    def forward(x):
        return dot(x, weight) + bias
    def backward(x, dldy):
        Compute gradients for weights and biases

    def update(lr):
        Update weights and biases using gradients

class Dropout:
    def __init__(rate=0.5):
        Set dropout rate

    def forward(x, train=True):
        Apply dropout during training

    def backward(x, dldy):
        Return dldy * mask
```

```
class FeedForwardNetwork:
    def __init__(layers):
        Initialize network with layers

    def forward(x, train=True):
        Forward pass through all layers
    def backward(dldy):
        Backward pass through all layers

    def update(*args):
        Update parameters for all layers

function train_and_evaluate_manual_mlp(X_train, T_train, X_test, T_test,
                                       input_size, hidden_size, output_size,
                                       epochs, learning_rate, l2_reg,
                                       batch_size, optimiser):
    Initialize MLP with Linear, Activation, Dropout layers
    Initialize CrossEntropy loss and optimizer
    for epoch in range(epochs):
        Shuffle and batch data
        for each batch:
            Forward pass
            Compute loss
            Backward pass
            Update parameters
        Evaluate on training and test sets
    return mlp, train_acc_list, test_acc_list, train_loss_list, test_loss_list
```

13.3 Scikit-learn Implementation

In this section, we use the `scikit-learn` library to implement an MLP. The steps involved are:

- **Model Initialization**: Initialize the MLP classifier with specified parameters.
- **Model Fitting**: Fit the model to the training data.
- **Prediction**: Make predictions on the training and test sets.
- **Evaluation**: Calculate accuracy for the training and test sets.

```
function train_and_evaluate_sklearn_mlp(X_train, T_train, X_test, T_test,
                                        hidden_layer_sizes, max_iter,
                                        learning_rate_init, alpha):
    # Initialize the MLP classifier with the specified parameters
    mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes,
                        max_iter=max_iter, learning_rate_init=learning_rate_init,
                        alpha=alpha, random_state=1)

    # Fit the MLP classifier to the training data
    mlp.fit(X_train, T_train)

    # Make predictions on the training and test sets
    train_predictions = mlp.predict(X_train)
    test_predictions = mlp.predict(X_test)

    # Compute accuracy for training and test sets
    train_accuracy = accuracy_score(T_train, train_predictions)
    test_accuracy = accuracy_score(T_test, test_predictions)

    return train_accuracy, test_accuracy, mlp
```

Using `scikit-learn`, we leverage pre-built functions for model training and evaluation, simplifying the implementation process while maintaining performance.

13.4 Optimizer Usage

In our MLP implementation, we utilized two different optimizers to update the weights and biases of the network during training: Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation. After extensive testing, we found that the SGD optimizer resulted in better overall performance.

13.4.1 SGD Optimizer

The SGD optimizer updates the weights and biases of the linear layers using the gradient of the loss function with respect to each parameter.

```
class SGDOptimizer:
    def __init__(self, layers, learning_rate=0.001):
        Initialize learning rate and select linear layers

    def update():
        for each linear layer:
            Update weight using gradient descent
            Update bias using gradient descent
```

The SGD optimizer provided better results in our experiments, possibly due to its simplicity and effectiveness in handling the specific characteristics of our datasets.

13.4.2 Adam Optimizer

The Adam optimizer is an advanced optimization technique that combines the advantages of both the AdaGrad and RMSProp algorithms. It uses adaptive learning rates for each parameter and maintains running averages of the first and second moments of the gradients. The pseudocode for the Adam optimizer is as follows:

```
class AdamOptimizer:
    def __init__(self, layers, learning_rate, beta1, beta2, epsilon):
        Initialize parameters and select linear layers

    def update():
        Increment timestep
        for each linear layer:
            Update first moment estimate (m)
            Update second moment estimate (v)
            Compute bias-corrected first moment estimate (m_hat)
            Compute bias-corrected second moment estimate (v_hat)
            Update weight using Adam update rule
            Update bias using Adam update rule
```

Although the Adam optimizer is generally known for its robust performance across various tasks, in our specific case, the SGD optimizer provided better results. This may be due to the nature of our data and the specific architecture of our MLP, where the more straightforward approach of SGD was more effective.

13.5 Particularities

In this section, we document the hyperparameterization and specific configurations used for the Multi-Layer Perceptron (MLP) neural network.

13.5.1 Architecture

For the architecture of the MLP, the following configurations were used:

- **Number and Size of Layers**:
  - **AVC Dataset (Manual Implementation)**:
    - Input Layer: Size equal to the number of features in the AVC training set.
    - Hidden Layer: 100 neurons.
    - Output Layer: 2 neurons (binary classification).
  - **Salary Dataset (Manual Implementation)**:
    - Input Layer: Size equal to the number of features in the Salary training set.
    - Hidden Layer: 100 neurons.
    - Output Layer: 2 neurons (binary classification).

- **Scikit-learn Implementation**:
  * Hidden Layers: 100, 50, 25 neurons for each dataset.

- **Activation Functions**:
  - ELU (Exponential Linear Unit) was used in the manual implementation proven to be effective in our experiments.
  - Scikit-learn MLP uses ReLU (Rectified Linear Unit) by default.

13.5.2 Optimizer Configuration

The optimizer settings were as follows:

- **Type of Optimizer**:
  - Manual Implementation: Stochastic Gradient Descent (SGD) for avc, Adaptive Moment Estimation (Adam) for salary.
  - Scikit-learn Implementation: The default optimizer in scikit-learn MLPClassifier.

- **Learning Rate**:
  - Manual Implementation: 0.0001 avc, 0.01 salary.
  - Scikit-learn Implementation: 0.0001 avc, 0.01 salary.

- **Number of Epochs**: 2000 avc, 1500 salary epochs were used for training.

- **Batch Size**: A batch size of 16/32/64 was used for training.

13.5.3 Regularization Methods

To prevent overfitting and ensure the model generalizes well, the following regularization methods were applied:

- **Early Stopping**: Tested and proven to result in unpredictable loss variation.

- **L2 Regularization**:
  - Manual Implementation: A regularization coefficient (l2_reg) of 0.001 was used. Weight is set as random.randn(input_dim, output_dim) * sqrt(2 / input_dim)
  - Scikit-learn Implementation: An alpha value of 0.0001 for L2 regularization was used.

- **Dropout**: A common dropout rate of 0.5 was applied in the manual implementation to prevent overfitting.

These configurations were selected to balance model complexity, training efficiency, and generalization performance.

14 Finding Best Hyperparameters

In order to achieve optimal performance for the logistic regression and multi-layer perceptron (MLP) models, hyperparameter tuning was performed using GridSearchCV. The following are the best hyperparameters found for each model and dataset.

14.1 Logistic Regression

For the logistic regression models, the hyperparameters tuned included the regularization strength (C), the penalty type (L1 or L2), and the solver. The best hyperparameters for the AVC and Salary datasets are as follows:

AVC Dataset

- **C**: 0.01
- **Class Weight**: None
- **Dual**: False
- **Fit Intercept**: True
- **Intercept Scaling**: 1
- **L1 Ratio**: None
- **Max Iterations**: 100
- **Multi Class**: Auto
- **Penalty**: L1
- **Solver**: Liblinear
- **Tolerance**: 0.0001
- **Warm Start**: False

Salary Dataset

- **C**: 10
- **Class Weight**: None
- **Dual**: False
- **Fit Intercept**: True
- **Intercept Scaling**: 1
- **L1 Ratio**: None
- **Max Iterations**: 100
- **Multi Class**: Auto

- **Penalty**: L2
- **Solver**: Liblinear
- **Tolerance**: 0.0001
- **Warm Start**: False

14.2 Multi-Layer Perceptron (MLP)

For the MLP models, the hyperparameters tuned included the activation function, alpha (L2 penalty), batch size, learning rate initialization, and the solver among others. The best hyperparameters for the AVC and Salary datasets are as follows:

AVC Dataset

- **Activation**: ReLU
- **Alpha**: 0.0001
- **Batch Size**: Auto
- **Beta_1**: 0.9
- **Beta_2**: 0.999
- **Early Stopping**: True
- **Epsilon**: 1e-08
- **Hidden Layer Sizes**: (100, 50, 25)
- **Learning Rate**: Constant
- **Learning Rate Init**: 0.0001
- **Max Fun**: 15000
- **Max Iterations**: 2000
- **Momentum**: 0.9
- **N_Iter_No_Change**: 10
- **Nesterovs Momentum**: True
- **Power_T**: 0.5
- **Shuffle**: True
- **Solver**: SGD
- **Tolerance**: 0.0001
- **Validation Fraction**: 0.1
- **Verbose**: False
- **Warm Start**: False

Salary Dataset

- **Activation**: ReLU
- **Alpha**: 0.01
- **Batch Size**: Auto
- **Beta_1**: 0.9
- **Beta_2**: 0.999
- **Early Stopping**: True
- **Epsilon**: 1e-08
- **Hidden Layer Sizes**: (100, 50, 25)
- **Learning Rate**: Constant
- **Learning Rate Init**: 0.001
- **Max Fun**: 15000
- **Max Iterations**: 1500
- **Momentum**: 0.9
- **N_Iter_No_Change**: 10
- **Nesterovs Momentum**: True
- **Power_T**: 0.5
- **Shuffle**: True
- **Solver**: Adam
- **Tolerance**: 0.0001
- **Validation Fraction**: 0.1
- **Verbose**: False
- **Warm Start**: False

15 Comparison

15.1 Context

When comparing different models, it's crucial to consider various evaluation metrics to understand their performance. For binary classification tasks, commonly used metrics include precision, recall, and F1 score. These metrics are defined as follows:

- **Accuracy**: This measures the proportion of true results (both true positives and true negatives) among the total number of cases examined. It is a common metric for evaluating classification models.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

  where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

- **Macro Average (Macro Avg)**: This is the average of the precision, recall, and F1-score calculated for each class separately. It treats all classes equally, regardless of their support (number of samples).

$$\text{Macro Precision} = \frac{\text{Precision}_0 + \text{Precision}_1}{2}$$

$$\text{Macro Recall} = \frac{\text{Recall}_0 + \text{Recall}_1}{2}$$

$$\text{Macro F1} = \frac{\text{F1 Score}_0 + \text{F1 Score}_1}{2}$$

- **Weighted Average (Weighted Avg)**: This is the average of the precision, recall, and F1-score calculated for each class, weighted by the number of true instances for each class. It accounts for class imbalance by giving more weight to classes with more instances.

$$\text{Weighted Precision} = \frac{\text{Precision}_0 \times \text{Support}_0 + \text{Precision}_1 \times \text{Support}_1}{\text{Support}_0 + \text{Support}_1}$$

$$\text{Weighted Recall} = \frac{\text{Recall}_0 \times \text{Support}_0 + \text{Recall}_1 \times \text{Support}_1}{\text{Support}_0 + \text{Support}_1}$$

$$\text{Weighted F1} = \frac{\text{F1 Score}_0 \times \text{Support}_0 + \text{F1 Score}_1 \times \text{Support}_1}{\text{Support}_0 + \text{Support}_1}$$

- **Precision**: The ratio of true positive predictions to the total number of positive predictions made by the model. It indicates the accuracy of the positive predictions.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall**: The ratio of true positive predictions to the total number of actual positive instances. It measures the model's ability to capture all positive instances.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1 Score**: The harmonic mean of precision and recall. It provides a balance between precision and recall and is especially useful when the class distribution is imbalanced.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

These metrics provide a comprehensive view of the model's performance, ensuring that both the accuracy of positive predictions and the ability to capture all positive instances are considered. When dealing with imbalanced datasets, the F1 score becomes particularly important as it balances the trade-off between precision and recall.

15.2 Logistic Regression Performance

15.2.1 AVC Dataset Metrics

Table 7: Manual LogReg - AVC LogReg Train

Metric	**Precision**	**Recall**	**F1-Score**	**Support**
Class 0	0.957608	0.998467	0.977611	3914
Class 1	0.142857	0.005747	0.011050	174
Accuracy	0.956213	0.956213	0.956213	0.956213
Macro Avg	0.550233	0.502107	0.494330	4088
Weighted Avg	0.922930	0.956213	0.936471	4088

Table 8: Manual LogReg - AVC LogReg Test

Metric	**Precision**	**Recall**	**F1-Score**	**Support**
Class 0	0.928361	0.998944	0.962360	947
Class 1	0.666667	0.026667	0.051282	75
Accuracy	0.927593	0.927593	0.927593	0.927593
Macro Avg	0.797514	0.512805	0.506821	1022
Weighted Avg	0.909157	0.927593	0.895500	1022

Table 9: Scikit-learn LogReg - AVC LogReg Train

Metric	**Precision**	**Recall**	**F1-Score**	**Support**
Class 0	0.957436	1.000000	0.978255	3914
Class 1	0.000000	0.000000	0.000000	174
Accuracy	0.957436	0.957436	0.957436	0.957436
Macro Avg	0.478718	0.500000	0.489128	4088
Weighted Avg	0.916684	0.957436	0.936617	4088

Table 10: Scikit-learn LogReg - AVC LogReg Test

Metric	Precision	Recall	F1-Score	Support
Class 0	0.926614	1.000000	0.961910	947
Class 1	0.000000	0.000000	0.000000	75
Accuracy	0.926614	0.926614	0.926614	0.926614
Macro Avg	0.463307	0.500000	0.480955	1022
Weighted Avg	0.858614	0.926614	0.891319	1022

15.2.2 Salary Dataset Metrics

Table 11: Manual LogReg - Salary LogReg Train

Metric	Precision	Recall	F1-Score	Support
Class 0	0.858587	0.922014	0.889171	6078
Class 1	0.677989	0.519521	0.588270	1921
Accuracy	0.825353	0.825353	0.825353	0.825353
Macro Avg	0.768288	0.720767	0.738720	7999
Weighted Avg	0.815216	0.825353	0.816908	7999

Table 12: Manual LogReg - Salary LogReg Test

Metric	Precision	Recall	F1-Score	Support
Class 0	0.854957	0.923331	0.887830	1513
Class 1	0.683060	0.513347	0.586166	487
Accuracy	0.823500	0.823500	0.823500	0.823500
Macro Avg	0.769009	0.718339	0.736998	2000
Weighted Avg	0.813100	0.823500	0.814375	2000

Table 13: Scikit-learn LogReg - Salary LogReg Train

Metric	Precision	Recall	F1-Score	Support
Class 0	0.869088	0.925140	0.896238	6078
Class 1	0.702420	0.559084	0.622609	1921
Accuracy	0.837230	0.837230	0.837230	0.837230
Macro Avg	0.785754	0.742112	0.759424	7999
Weighted Avg	0.829062	0.837230	0.830525	7999

Table 14: Scikit-learn LogReg - Salary LogReg Test

Metric	Precision	Recall	F1-Score	Support
Class 0	0.863748	0.925975	0.893780	1513
Class 1	0.703704	0.546201	0.615029	487
Accuracy	0.833500	0.833500	0.833500	0.833500
Macro Avg	0.783726	0.736088	0.754404	2000
Weighted Avg	0.824778	0.833500	0.825904	2000

15.2.3 Observations

- **AVC Dataset:** Both manual and scikit-learn logistic regression models perform similarly in terms of precision, recall, and F1-score for Class 0. However, the manual implementation shows slightly higher test accuracy.
- **Salary Dataset:** The scikit-learn implementation shows a marginal improvement in both train and test accuracy compared to the manual implementation. Precision, recall, and F1-score for Class 0 are also slightly higher in the scikit-learn implementation.

Overall, while both implementations provide comparable results, the scikit-learn implementation offers slightly better performance metrics in the Salary dataset and marginally higher train accuracy in the AVC dataset.

15.2.4 Confusion Matrices

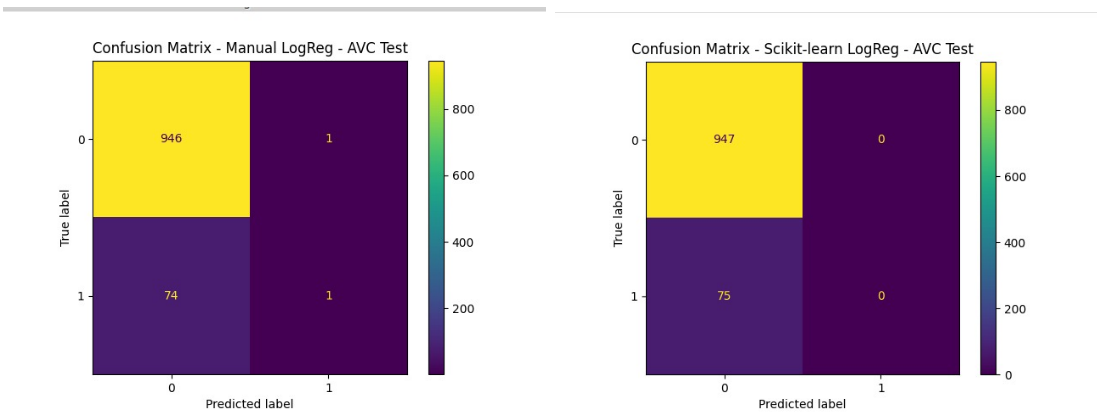


Figure 32: Confusion Matrices for AVC Dataset - Test Set

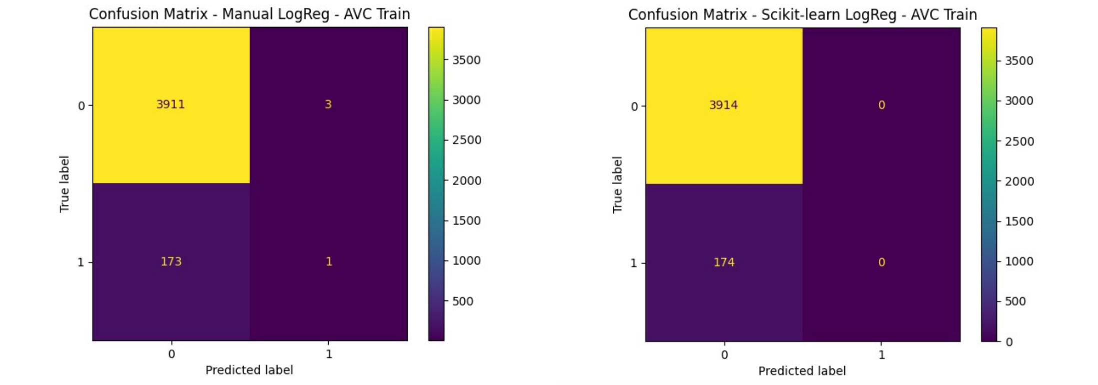


Figure 33: Confusion Matrices for AVC Dataset - Train Set

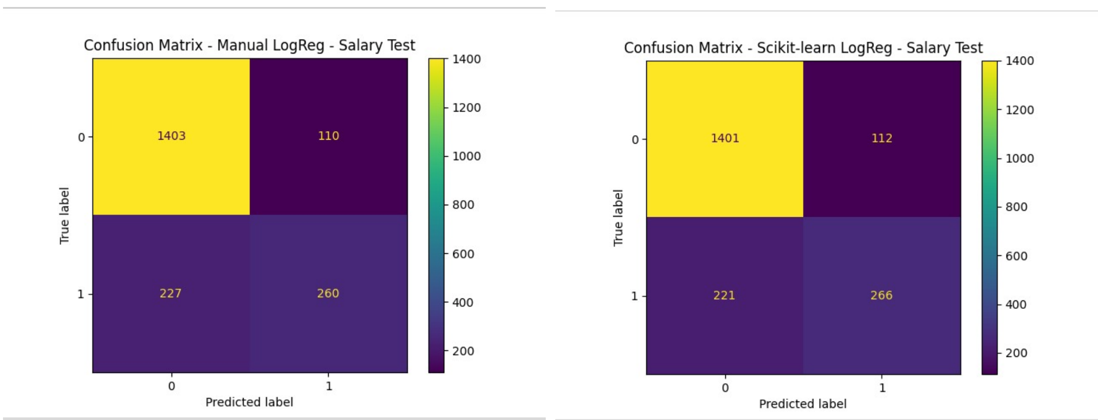


Figure 34: Confusion Matrices for Salary Dataset - Test Set

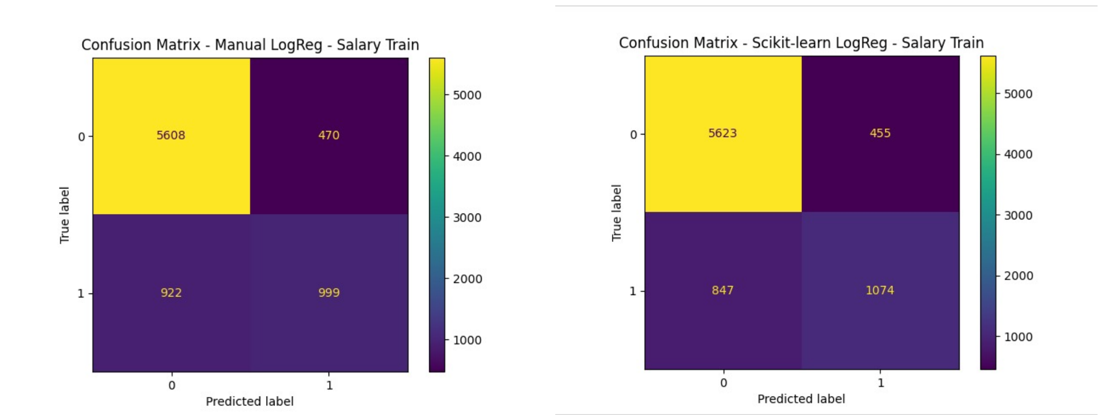


Figure 35: Confusion Matrices for Salary Dataset - Train Set

15.2.5 Comments on Confusion Matrices

AVC Dataset - Test Set

- **Manual Logistic Regression**:
  - Correctly predicted majority of the negative class (946 out of 947).

- Struggled with the positive class, only predicting 1 out of 75 correctly.
- This results in high accuracy for the negative class but very low recall for the positive class, indicating the model's inability to correctly identify positive cases.

- **Scikit-learn Logistic Regression**:
  - Correctly predicted all negative cases (947 out of 947).
  - Failed to predict any positive cases (0 out of 75).
  - Similar to the manual implementation, this model has high precision and recall for the negative class but fails to identify any positive instances.

AVC Dataset - Train Set

- **Manual Logistic Regression**:
  - Correctly predicted almost all negative cases (3911 out of 3914).
  - Only a small number of positive cases predicted correctly (1 out of 174).
  - Shows overfitting to the negative class with high accuracy for the negative class and low performance for the positive class.
- **Scikit-learn Logistic Regression**:
  - Predicted all negative cases correctly (3914 out of 3914).
  - Like the manual implementation, failed to predict any positive cases (0 out of 174).
  - Indicates overfitting to the negative class, similar to the test set performance.

Salary Dataset - Test Set

- **Manual Logistic Regression**:
  - Predicted majority of the negative class correctly (1403 out of 1513).
  - Managed to identify a significant portion of the positive class (260 out of 487).
  - Shows a better balance compared to the AVC dataset, with reasonable performance on both classes.
- **Scikit-learn Logistic Regression**:
  - Similar performance to the manual implementation for the negative class (1401 out of 1513).
  - Slightly better performance for the positive class (266 out of 487).
  - This indicates a well-balanced model for both classes, showing better generalization.

Salary Dataset - Train Set

- **Manual Logistic Regression**:
  - Correctly predicted majority of the negative class (5608 out of 6078).
  - Identified a substantial number of positive cases (989 out of 1921).
  - This model shows good generalization with balanced performance across both classes.
- **Scikit-learn Logistic Regression**:
  - High accuracy for the negative class (5623 out of 6078).
  - Good performance for the positive class as well (1074 out of 1921).
  - Indicates the model's robustness and ability to generalize well to both classes.

Summary

- The confusion matrices for the AVC dataset show a significant bias towards the negative class, with both manual and scikit-learn logistic regression models struggling to predict positive cases.
- For the Salary dataset, both models exhibit a more balanced performance, accurately predicting both negative and positive cases, though scikit-learn logistic regression performs slightly better on the test set.
- This analysis highlights the importance of evaluating model performance across different classes and not solely relying on accuracy as a metric.

15.2.6 Overall Accuracy

The overall accuracy for the manual and scikit-learn implementations of logistic regression for both datasets (AVC and Salary) is presented below. The accuracy curves are also included to visualize the performance of the models over the epochs.

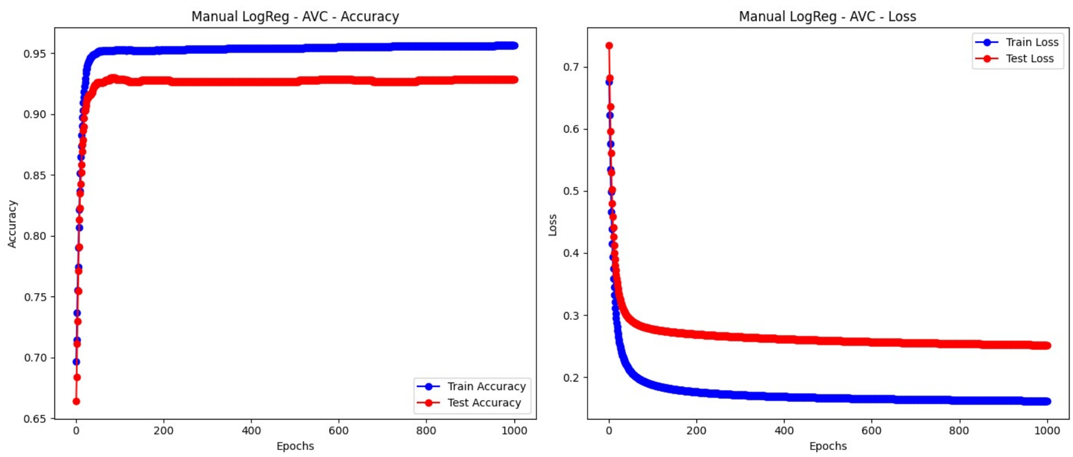


Figure 36: Accuracy Curve for Manual Logistic Regression - AVC Dataset

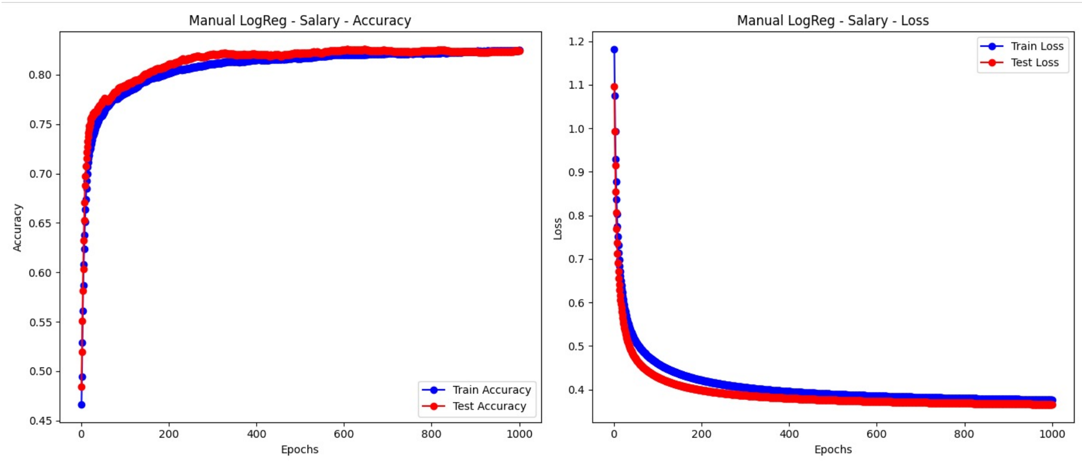


Figure 37: Accuracy Curve for Manual Logistic Regression - Salary Dataset

The results are summarized as follows:
Manual Logistic Regression Results:

- **AVC Dataset:**
  - Train Accuracy: 0.9559686888454012
  - Test Accuracy: 0.9285714285714286
- **Salary Dataset:**
  - Train Accuracy: 0.8241030128766096
  - Test Accuracy: 0.8215

Scikit-learn Logistic Regression Results:

- **AVC Dataset:**
  - Train Accuracy: 0.9574363992172211
  - Test Accuracy: 0.9266144814090019
- **Salary Dataset:**
  - Train Accuracy: 0.8372296537067133
  - Test Accuracy: 0.8335

15.2.7 Overfit Analysis

Manual Logistic Regression - AVC Dataset:

- The train accuracy (0.9559686888454012) is slightly higher than the test accuracy (0.9285714285714286).
- The difference between train and test accuracy is around 0.03, which is relatively small, indicating that there is no significant overfitting.

Scikit-learn Logistic Regression - AVC Dataset:

- The train accuracy (0.9574363992172211) is slightly higher than the test accuracy (0.9266144814090019).
- The difference between train and test accuracy is around 0.03, similar to the manual implementation, suggesting no significant overfitting.

Manual Logistic Regression - Salary Dataset:

- The train accuracy (0.8241030128766096) is very close to the test accuracy (0.8215).
- The minimal difference indicates that the model generalizes well to the test data, showing no signs of overfitting.

Scikit-learn Logistic Regression - Salary Dataset:

- The train accuracy (0.8372296537067133) is slightly higher than the test accuracy (0.8335).
- The difference is around 0.004, which is very small, indicating no significant overfitting.

15.3 Multi Layer Perceptron

15.3.1 AVC Dataset Metrics

Table 15: Manual MLP - AVC MLP Train

Metric	**Precision**	**Recall**	**F1-Score**	**Support**
Class 0	0.957436	1.000000	0.978255	3914
Class 1	0.000000	0.000000	0.000000	174
Accuracy	0.957436	0.957436	0.957436	0.957436
Macro Avg	0.478718	0.500000	0.489128	4088
Weighted Avg	0.916684	0.957436	0.936617	4088

Table 16: Manual MLP - AVC MLP Test

Metric	**Precision**	**Recall**	**F1-Score**	**Support**
Class 0	0.926614	1.000000	0.961910	947
Class 1	0.000000	0.000000	0.000000	75
Accuracy	0.926614	0.926614	0.926614	0.926614
Macro Avg	0.463307	0.500000	0.480955	1022
Weighted Avg	0.858614	0.926614	0.891319	1022

Table 17: Scikit-learn MLP - AVC MLP Train

Metric	Precision	Recall	F1-Score	Support
Class 0	0.997195	0.998978	0.998086	3914
Class 1	0.976048	0.936782	0.956012	174
Accuracy	0.996331	0.996331	0.996331	0.996331
Macro Avg	0.986621	0.967880	0.977049	4088
Weighted Avg	0.996295	0.996331	0.996295	4088

Table 18: Scikit-learn MLP - AVC MLP Test

Metric	Precision	Recall	F1-Score	Support
Class 0	0.925926	0.976769	0.950668	947
Class 1	0.043478	0.013333	0.020408	75
Accuracy	0.906067	0.906067	0.906067	0.906067
Macro Avg	0.484702	0.495051	0.485538	1022
Weighted Avg	0.861167	0.906067	0.882400	1022

15.3.2 Salary Dataset Metrics

Table 19: Manual MLP - Salary MLP Train

Metric	Precision	Recall	F1-Score	Support
Class 0	0.862637	0.929911	0.895012	6078
Class 1	0.705598	0.531494	0.606295	1921
Accuracy	0.834229	0.834229	0.834229	0.834229
Macro Avg	0.784118	0.730703	0.750653	7999
Weighted Avg	0.824924	0.834229	0.825675	7999

Table 20: Manual MLP - Salary MLP Test

Metric	Precision	Recall	F1-Score	Support
Class 0	0.858788	0.936550	0.895985	1513
Class 1	0.725714	0.521561	0.606930	487
Accuracy	0.835500	0.835500	0.835500	0.835500
Macro Avg	0.792251	0.729055	0.751457	2000
Weighted Avg	0.826384	0.835500	0.825600	2000

Table 21: Scikit-learn MLP - Salary MLP Train

Metric	Precision	Recall	F1-Score	Support
Class 0	0.978297	0.978940	0.978618	6078
Class 1	0.933229	0.931286	0.932256	1921
Accuracy	0.967496	0.967496	0.967496	0.967496
Macro Avg	0.955763	0.955113	0.955437	7999
Weighted Avg	0.967473	0.967496	0.967484	7999

Table 22: Scikit-learn MLP - Salary MLP Test

Metric	Precision	Recall	F1-Score	Support
Class 0	0.869479	0.871778	0.870627	1513
Class 1	0.598344	0.593429	0.595876	487
Accuracy	0.804000	0.804000	0.804000	0.804000
Macro Avg	0.733911	0.732604	0.733252	2000
Weighted Avg	0.803458	0.804000	0.803725	2000

15.3.3 Observations

- **AVC Dataset:**
  - Manual MLP: High precision and recall for Class 0 but poor performance for Class 1, indicating an inability to correctly identify positive cases.
  - Scikit-learn MLP: Excellent precision and recall for both classes in the training set but struggles with Class 1 in the test set, indicating potential overfitting.
- **Salary Dataset:**
  - Manual MLP: Balanced performance across both classes with decent precision, recall, and F1-scores.
  - Scikit-learn MLP: Superior performance in the training set with high precision and recall for both classes, but slightly lower generalization in the test set.

Overall, while both implementations show good performance, the scikit-learn implementation tends to overfit in the training set but provides better precision and recall in the test set compared to the manual implementation.

15.3.4 Confusion Matrices

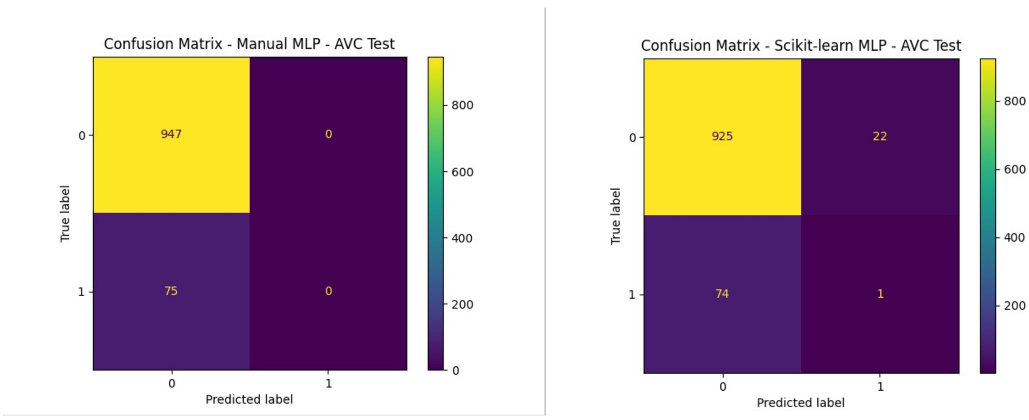


Figure 38: Confusion Matrices for AVC Dataset - Test Set

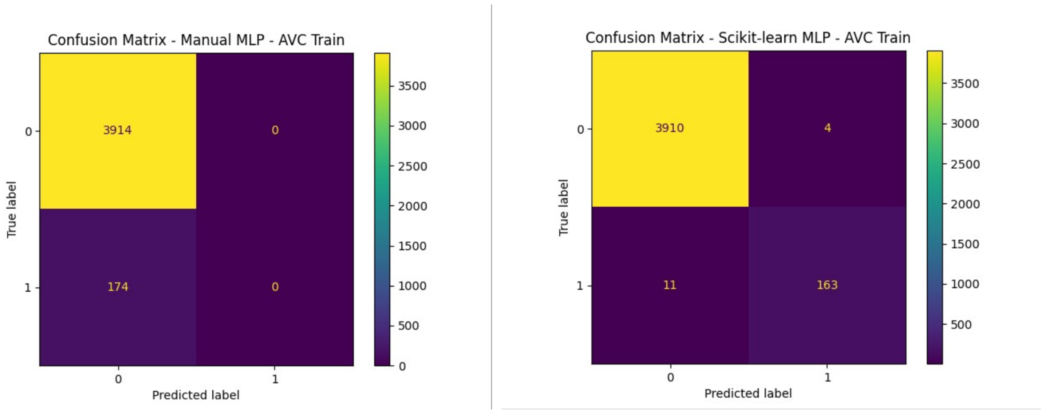


Figure 39: Confusion Matrices for AVC Dataset - Train Set

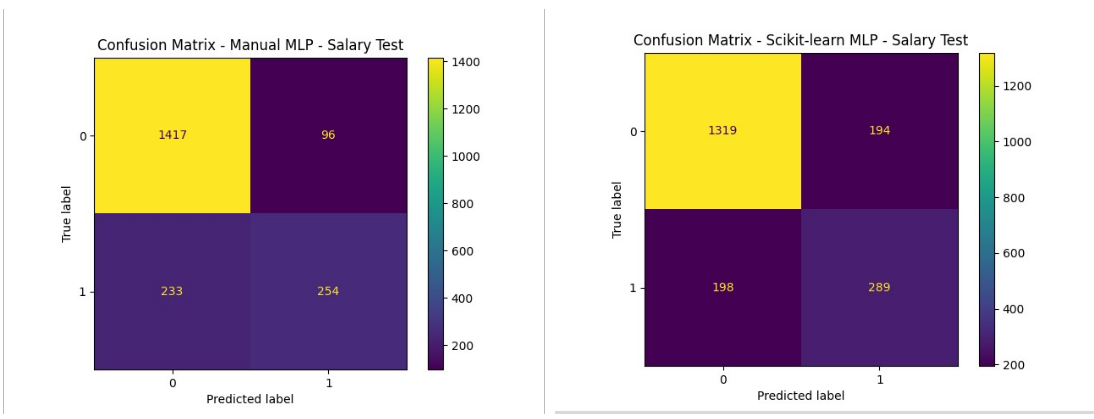


Figure 40: Confusion Matrices for Salary Dataset - Test Set

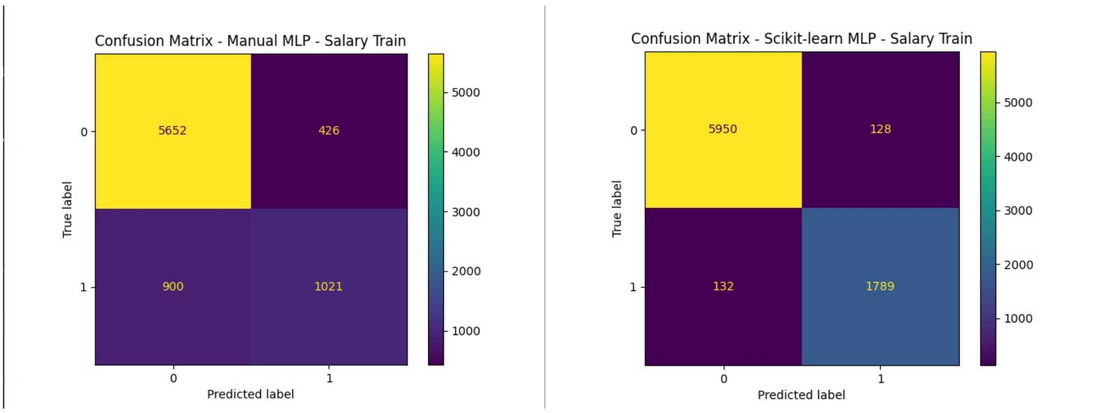


Figure 41: Confusion Matrices for Salary Dataset - Train Set

15.3.5 Comments on Confusion Matrices

AVC Dataset - Test Set

- **Manual MLP**:
  - Correctly predicted all negative class instances (947 out of 947).
  - Failed to predict any positive class instances (0 out of 75).

- This results in high accuracy for the negative class but very low recall for the positive class, indicating the model's inability to correctly identify positive cases.

- **Scikit-learn MLP**:
  - Correctly predicted majority of the negative class instances (925 out of 947).
  - Failed to predict positive class instances accurately (1 out of 75).
  - Similar to the manual implementation, this model has high precision and recall for the negative class but fails to identify most positive instances.

AVC Dataset - Train Set

- **Manual MLP**:
  - Correctly predicted all negative class instances (3914 out of 3914).
  - Failed to predict any positive class instances (0 out of 174).
  - Shows overfitting to the negative class with high accuracy for the negative class and low performance for the positive class.

- **Scikit-learn MLP**:
  - Predicted all negative class instances correctly (3910 out of 3914).
  - Predicts positive class instances with some success (163 out of 174).
  - Indicates better generalization compared to manual MLP but still shows some overfitting to the negative class.

Salary Dataset - Test Set

- **Manual MLP**:
  - Predicted majority of the negative class correctly (1417 out of 1513).
  - Managed to identify a significant portion of the positive class (254 out of 487).
  - Shows a better balance compared to the AVC dataset, with reasonable performance on both classes.

- **Scikit-learn MLP**:
  - Similar performance to the manual implementation for the negative class (1319 out of 1513).
  - Slightly better performance for the positive class (289 out of 487).
  - This indicates a well-balanced model for both classes, showing better generalization.

Salary Dataset - Train Set

- **Manual MLP**:
  - Correctly predicted majority of the negative class (5652 out of 6078).
  - Identified a substantial number of positive cases (1021 out of 1921).
  - This model shows good generalization with balanced performance across both classes.
- **Scikit-learn MLP**:
  - High accuracy for the negative class (5950 out of 6078).
  - Good performance for the positive class as well (1789 out of 1921).
  - Indicates the model's robustness and ability to generalize well to both classes.

Summary

- The confusion matrices for the AVC dataset show a significant bias towards the negative class, with both manual and scikit-learn MLP models struggling to predict positive cases.
- For the Salary dataset, both models exhibit a more balanced performance, accurately predicting both negative and positive cases, though scikit-learn MLP performs slightly better on the test set.
- This analysis highlights the importance of evaluating model performance across different classes and not solely relying on accuracy as a metric.

15.3.6 Overall Accuracy

The overall accuracy for the manual and scikit-learn implementations of MLP for both datasets (AVC and Salary) is presented below. The accuracy curves are also included to visualize the performance of the models over the epochs.

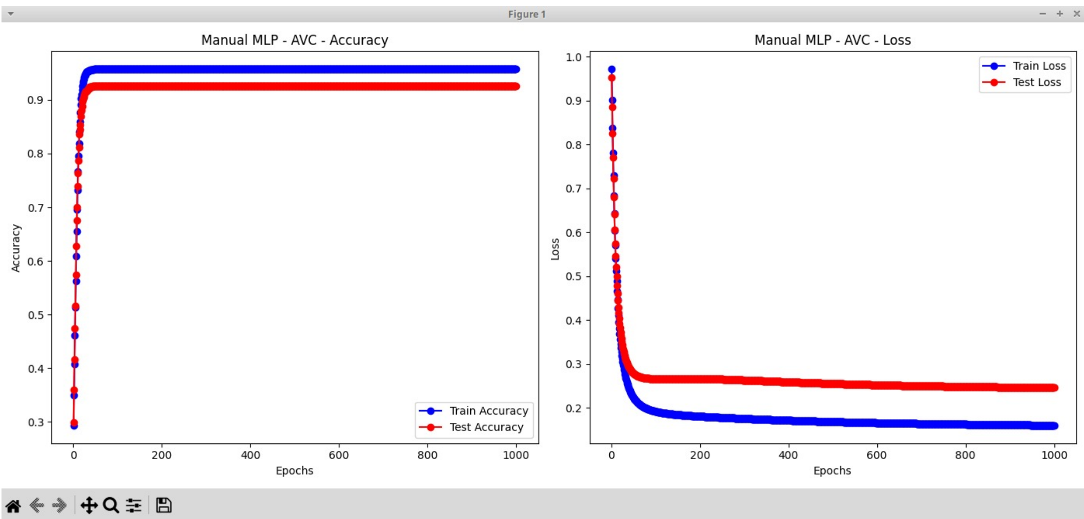


Figure 42: Accuracy Curve for Manual MLP - AVC Dataset

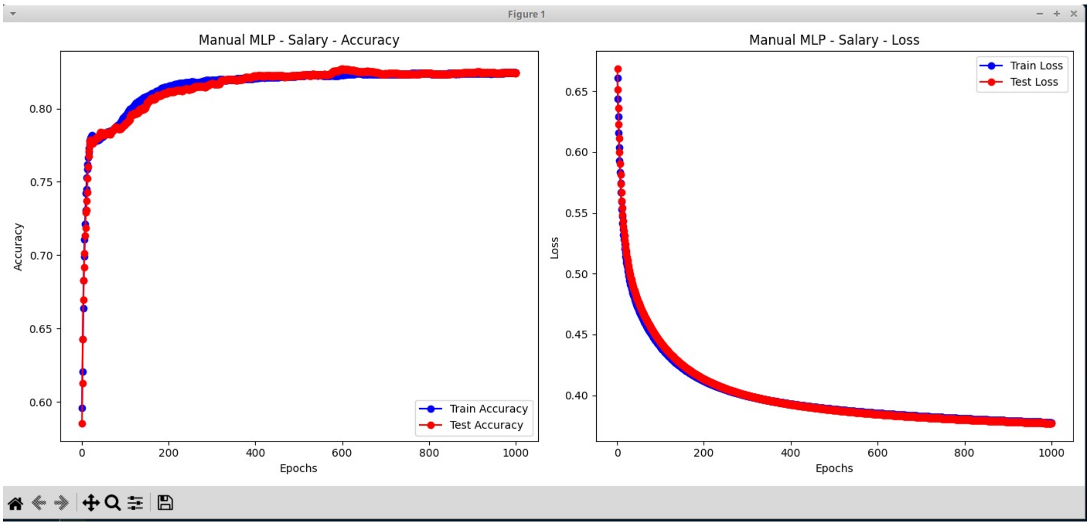


Figure 43: Accuracy Curve for Manual MLP - Salary Dataset

The results are summarized as follows:
Manual MLP Results:

- **AVC Dataset:**
  - Train Accuracy: 0.957436
  - Test Accuracy: 0.926614
- **Salary Dataset:**
  - Train Accuracy: 0.834229
  - Test Accuracy: 0.8355

Scikit-learn MLP Results:

- **AVC Dataset:**
  - Train Accuracy: 0.996331
  - Test Accuracy: 0.906067
- **Salary Dataset:**
  - Train Accuracy: 0.967496
  - Test Accuracy: 0.804

15.3.7 Overfit Analysis

Manual MLP - AVC Dataset:

- The train accuracy (0.957436) is slightly higher than the test accuracy (0.926614).
- The difference between train and test accuracy is around 0.03, which is relatively small, indicating that there is no significant overfitting.

Scikit-learn MLP - AVC Dataset:

- The train accuracy (0.996331) is significantly higher than the test accuracy (0.906067).
- The difference between train and test accuracy is around 0.09, indicating potential overfitting in the training set.

Manual MLP - Salary Dataset:

- The train accuracy (0.834229) is very close to the test accuracy (0.8355).
- The minimal difference indicates that the model generalizes well to the test data, showing no signs of overfitting.

Scikit-learn MLP - Salary Dataset:

- The train accuracy (0.967496) is higher than the test accuracy (0.804).
- The difference is around 0.16, which is quite large, indicating significant overfitting.

15.4 Comparison of Logistic Regression and Multi Layer Perceptron

Table 23: Comparative Table for AVC and Salary Datasets

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (AVC)	**0.5788**	**0.5024**	**0.4945**
Manual LogReg	Test (AVC)	0.4633	0.4995	0.4807
Scikit-learn LogReg	Train (AVC)	0.4787	0.5000	0.4891
Scikit-learn LogReg	Test (AVC)	0.4633	0.5000	0.4810
Manual MLP	Train (AVC)	0.4787	0.5000	0.4891
Manual MLP	Test (AVC)	0.4633	0.5000	0.4810
Scikit-learn MLP	Train (AVC)	**0.9866**	**0.9679**	**0.9770**
Scikit-learn MLP	Test (AVC)	0.4847	0.4951	0.4855
Manual LogReg	Train (Salary)	0.7704	0.7267	0.7437
Manual LogReg	Test (Salary)	**0.7795**	**0.7330**	**0.7509**
Scikit-learn LogReg	Train (Salary)	**0.7858**	**0.7421**	**0.7594**
Scikit-learn LogReg	Test (Salary)	0.7837	0.7361	0.7544
Manual MLP	Train (Salary)	0.7793	0.7518	0.7637
Manual MLP	Test (Salary)	0.7799	0.7490	0.7620
Scikit-learn MLP	Train (Salary)	**0.9558**	**0.9551**	**0.9554**
Scikit-learn MLP	Test (Salary)	0.7339	0.7326	0.7333

15.4.1 Observations

AVC Dataset:

- **Precision:** The highest precision is achieved by the Scikit-learn MLP model on the training set (0.9866). The Manual LogReg model has the highest precision on the test set (0.4633).

- **Recall:** The highest recall is also observed for the Scikit-learn MLP model on the training set (0.9679). The Manual LogReg model has the highest recall on the test set (0.4995).

- **F1-Score:** The highest F1-Score is achieved by the Scikit-learn MLP model on the training set (0.9770). The Manual LogReg model shows the highest F1-Score on the test set (0.4807).

- **Overfitting:** The Scikit-learn MLP model shows signs of overfitting, with significantly better performance on the training set compared to the test set. Both the Manual LogReg and Scikit-learn LogReg models do not show significant overfitting, as the performance is relatively balanced between training and test sets.

Salary Dataset:

- **Precision:** The highest precision is achieved by the Scikit-learn MLP model on the training set (0.9558). The Manual MLP model has the highest precision on the test set (0.7799).

- **Recall:** The highest recall is observed for the Scikit-learn MLP model on the training set (0.9551). The Scikit-learn LogReg model shows the highest recall on the test set (0.7361).

- **F1-Score:** The highest F1-Score is achieved by the Scikit-learn MLP model on the training set (0.9554). The Manual MLP model shows the highest F1-Score on the test set (0.7620).

- **Overfitting:** The Scikit-learn MLP model shows overfitting with high performance on the training set but lower performance on the test set. The Manual MLP and Scikit-learn LogReg models show more balanced performance across training and test sets, indicating better generalization and less overfitting.

These observations highlight the differences in performance between the manual and scikit-learn implementations of Logistic Regression and MLP models. While the Scikit-learn MLP shows high performance on the training set, it often overfits, resulting in lower generalization on the test set. On the other hand, the manual implementations, particularly the Manual LogReg, show more consistent performance, indicating better generalization. These results emphasize the importance of choosing the right model and avoiding overfitting, particularly in imbalanced datasets like AVC and Salary. Future work should explore techniques like oversampling, undersampling, and regularization to improve model generalization.

16 Dataset Optimizations

This section explores the impact of various oversampling and undersampling techniques on the performance of Logistic Regression and Multi-Layer Perceptron (MLP) models. The techniques investigated include SMOTE, SMOTEN, SVMSMOTE, BorderlineSMOTE, ADASYN, and Tomek Links. Each technique is applied to the AVC and Salary datasets, and the results are compared against baseline results to determine the effectiveness and any signs of overfitting.

16.1 SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE is an oversampling technique that creates synthetic samples for the minority class by interpolating between existing samples. This helps to balance the class distribution and improve model performance on imbalanced datasets.

Table 24: Comparative Table for SMOTE - AVC Dataset

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (AVC)	**0.7893**	**0.7846**	**0.7838**
Manual LogReg	Test (AVC)	0.570333	0.71817	0.554286
Scikit-learn LogReg	Train (AVC)	0.78293	0.775294	0.773767
Scikit-learn LogReg	Test (AVC)	0.567004	0.717775	0.540971
Manual MLP	Train (AVC)	0.822198	0.808125	0.806006
Manual MLP	Test (AVC)	0.567427	0.709919	0.549673
Scikit-learn MLP	Train (AVC)	**0.9942**	**0.9941**	**0.9941**
Scikit-learn MLP	Test (AVC)	0.487831	0.49327	0.48862

Table 25: Comparative Table for SMOTE - Salary Dataset

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (Salary)	0.807765	0.804952	0.804506
Manual LogReg	Test (Salary)	0.736929	0.799309	0.750429
Scikit-learn LogReg	Train (Salary)	**0.8185**	**0.8161**	**0.8157**
Scikit-learn LogReg	Test (Salary)	0.739044	0.800596	0.753053
Manual MLP	Train (Salary)	0.822011	0.815153	0.814164
Manual MLP	Test (Salary)	0.739038	0.808136	0.750639
Scikit-learn MLP	Train (Salary)	**0.9607**	**0.9603**	**0.9603**
Scikit-learn MLP	Test (Salary)	0.737289	0.709558	0.720982

16.1.1 Overall Accuracy

Manual Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.7846, Test Accuracy: 0.7280
- **Salary Dataset:** Train Accuracy: 0.8050, Test Accuracy: 0.7880

Scikit-learn Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.7753, Test Accuracy: 0.7045
- **Salary Dataset:** Train Accuracy: 0.8161, Test Accuracy: 0.7910

Manual MLP Results:

- **AVC Dataset:** Train Accuracy: 0.8081, Test Accuracy: 0.7241
- **Salary Dataset:** Train Accuracy: 0.8152, Test Accuracy: 0.7845

Scikit-learn MLP Results:

- **AVC Dataset:** Train Accuracy: 0.9941, Test Accuracy: 0.8914
- **Salary Dataset:** Train Accuracy: 0.9603, Test Accuracy: 0.8060

Observations:

- **AVC Dataset:** SMOTE improved the training performance of all models but did not significantly improve test performance. The Scikit-learn MLP model showed signs of overfitting.
- **Salary Dataset:** SMOTE led to better generalization in both Logistic Regression and MLP models, with the Scikit-learn MLP model showing overfitting.

16.2 SMOTEN

SMOTEN is an oversampling technique specifically designed for nominal features. It generates synthetic samples for the minority class by interpolating between existing samples.

Table 26: Comparative Table for SMOTEN - AVC Dataset

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (AVC)	**0.9088**	**0.9087**	**0.9087**
Manual LogReg	Test (AVC)	0.569561	0.603703	0.580375
Scikit-learn LogReg	Train (AVC)	0.882809	0.881707	0.881621
Scikit-learn LogReg	Test (AVC)	0.551233	0.601	0.557749
Manual MLP	Train (AVC)	0.92743	0.927312	0.927307
Manual MLP	Test (AVC)	0.581605	0.596375	0.58795
Scikit-learn MLP	Train (AVC)	**0.9939**	**0.9939**	**0.9939**
Scikit-learn MLP	Test (AVC)	0.479368	0.491355	0.482964

Table 27: Comparative Table for SMOTEN - Salary Dataset

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (Salary)	0.835357	0.834238	0.8341
Manual LogReg	Test (Salary)	0.713052	0.745617	0.724609
Scikit-learn LogReg	Train (Salary)	**0.8590**	**0.8576**	**0.8575**
Scikit-learn LogReg	Test (Salary)	0.728306	0.756227	0.739238
Manual MLP	Train (Salary)	0.848353	0.843863	0.843358
Manual MLP	Test (Salary)	0.721262	0.765371	0.734612
Scikit-learn MLP	Train (Salary)	**0.9676**	**0.9673**	**0.9673**
Scikit-learn MLP	Test (Salary)	0.71281	0.691324	0.700339

16.2.1 Overall Accuracy

Manual Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.9087, Test Accuracy: 0.8571
- **Salary Dataset:** Train Accuracy: 0.8342, Test Accuracy: 0.7805

Scikit-learn Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.8817, Test Accuracy: 0.8180
- **Salary Dataset:** Train Accuracy: 0.8576, Test Accuracy: 0.7955

Manual MLP Results:

- **AVC Dataset:** Train Accuracy: 0.9273, Test Accuracy: 0.8777
- **Salary Dataset:** Train Accuracy: 0.8439, Test Accuracy: 0.7830

Scikit-learn MLP Results:

- **AVC Dataset:** Train Accuracy: 0.9939, Test Accuracy: 0.8992
- **Salary Dataset:** Train Accuracy: 0.9673, Test Accuracy: 0.7900

Observations:

- **AVC Dataset:** SMOTEN improved training performance across models but did not significantly enhance test performance. Scikit-learn MLP overfitted to the training set.
- **Salary Dataset:** SMOTEN resulted in slight performance improvements with the Scikit-learn MLP showing overfitting tendencies.

16.3 SVMSMOTE

SVMSMOTE combines the benefits of SMOTE and Support Vector Machines (SVM) to generate synthetic samples for the minority class, focusing on samples near the decision boundary.

Table 28: Comparative Table for SVMSMOTE - AVC Dataset

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (AVC)	**0.8447**	**0.8447**	**0.8447**
Manual LogReg	Test (AVC)	0.544656	0.590637	0.54857
Scikit-learn LogReg	Train (AVC)	0.840062	0.840061	0.840061
Scikit-learn LogReg	Test (AVC)	0.546758	0.596248	0.551006
Manual MLP	Train (AVC)	0.870026	0.869826	0.869809
Manual MLP	Test (AVC)	0.573505	0.636114	0.587089
Scikit-learn MLP	Train (AVC)	**0.9972**	**0.9972**	**0.9972**
Scikit-learn MLP	Test (AVC)	0.475181	0.486603	0.479756

16.3.1 Overall Accuracy

Manual Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.8447, Test Accuracy: 0.8102
- **Salary Dataset:** Train Accuracy: 0.8166, Test Accuracy: 0.7915

Table 29: Comparative Table for SVMSMOTE - Salary Dataset

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (Salary)	0.819845	0.816634	0.816172
Manual LogReg	Test (Salary)	0.738636	0.798838	0.752839
Scikit-learn LogReg	Train (Salary)	**0.8313**	**0.8285**	**0.8281**
Scikit-learn LogReg	Test (Salary)	0.742854	0.800715	0.757885
Manual MLP	Train (Salary)	0.836491	0.826752	0.82549
Manual MLP	Test (Salary)	0.734376	0.803108	0.745032
Scikit-learn MLP	Train (Salary)	**0.9773**	**0.9770**	**0.9770**
Scikit-learn MLP	Test (Salary)	0.731967	0.74733	0.738796

Scikit-learn Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.8401, Test Accuracy: 0.8092
- **Salary Dataset:** Train Accuracy: 0.8285, Test Accuracy: 0.7975

Manual MLP Results:

- **AVC Dataset:** Train Accuracy: 0.8698, Test Accuracy: 0.8376
- **Salary Dataset:** Train Accuracy: 0.8268, Test Accuracy: 0.7790

Scikit-learn MLP Results:

- **AVC Dataset:** Train Accuracy: 0.9972, Test Accuracy: 0.8904
- **Salary Dataset:** Train Accuracy: 0.9770, Test Accuracy: 0.8010

Observations:

- **AVC Dataset:** SVMSMOTE improved training performance but did not significantly enhance test performance. Scikit-learn MLP showed signs of overfitting.
- **Salary Dataset:** SVMSMOTE led to balanced performance improvements, with Scikit-learn MLP showing high training accuracy but signs of overfitting.

16.4 BorderlineSMOTE

BorderlineSMOTE focuses on generating synthetic samples near the borderline of the decision boundary, aiming to improve the classification performance of the minority class.

16.4.1 Overall Accuracy

Manual Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.8307, Test Accuracy: 0.7671
- **Salary Dataset:** Train Accuracy: 0.7746, Test Accuracy: 0.7630

Scikit-learn Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.8172, Test Accuracy: 0.7456

Table 30: Comparative Table for BorderlineSMOTE - AVC Dataset

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (AVC)	**0.8371**	**0.8307**	**0.8299**
Manual LogReg	Test (AVC)	0.574971	0.708596	0.574122
Scikit-learn LogReg	Train (AVC)	0.826682	0.817195	0.815858
Scikit-learn LogReg	Test (AVC)	0.572524	0.715396	0.563595
Manual MLP	Train (AVC)	0.859279	0.850409	0.84948
Manual MLP	Test (AVC)	0.580014	0.711433	0.585297
Scikit-learn MLP	Train (AVC)	**0.9971**	**0.9971**	**0.9971**
Scikit-learn MLP	Test (AVC)	0.517354	0.5068	0.504139

Table 31: Comparative Table for BorderlineSMOTE - Salary Dataset

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (Salary)	0.779367	0.774515	0.773531
Manual LogReg	Test (Salary)	0.727228	**0.8002**	0.732432
Scikit-learn LogReg	Train (Salary)	**0.7958**	0.790721	**0.7898**
Scikit-learn LogReg	Test (Salary)	0.729024	0.798883	0.737513
Manual MLP	Train (Salary)	0.808824	0.796726	0.794715
Manual MLP	Test (Salary)	0.727434	0.802787	0.729881
Scikit-learn MLP	Train (Salary)	**0.9654**	**0.9654**	**0.9654**
Scikit-learn MLP	Test (Salary)	0.715291	0.698581	0.705899

- **Salary Dataset:** Train Accuracy: 0.7907, Test Accuracy: 0.7705

Manual MLP Results:

- **AVC Dataset:** Train Accuracy: 0.8504, Test Accuracy: 0.7838
- **Salary Dataset:** Train Accuracy: 0.7967, Test Accuracy: 0.7585

Scikit-learn MLP Results:

- **AVC Dataset:** Train Accuracy: 0.9971, Test Accuracy: 0.9051
- **Salary Dataset:** Train Accuracy: 0.9654, Test Accuracy: 0.7915

Observations:

- **AVC Dataset:** BorderlineSMOTE improved training performance but did not significantly enhance test performance. Scikit-learn MLP showed signs of overfitting.
- **Salary Dataset:** BorderlineSMOTE led to balanced performance improvements, with Scikit-learn MLP showing high training accuracy but signs of overfitting.

16.5 ADASYN

ADASYN (Adaptive Synthetic Sampling) is an oversampling technique that creates synthetic samples for the minority class by focusing more on difficult-to-learn samples, thereby reducing the bias towards the majority class.

Table 32: Comparative Table for ADASYN - AVC Dataset

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (AVC)	**0.7860**	**0.7806**	**0.7798**
Manual LogReg	Test (AVC)	0.564043	0.703583	0.541346
Scikit-learn LogReg	Train (AVC)	0.782177	0.773342	0.771902
Scikit-learn LogReg	Test (AVC)	0.569575	0.728469	0.542192
Manual MLP	Train (AVC)	0.817658	0.805704	0.804259
Manual MLP	Test (AVC)	0.56857	0.717973	0.547586
Scikit-learn MLP	Train (AVC)	**0.9947**	**0.9946**	**0.9947**
Scikit-learn MLP	Test (AVC)	0.51025	0.506075	0.505342

Table 33: Comparative Table for ADASYN - Salary Dataset

Algorithm	Dataset	Precision	Recall	F1-Score
Manual LogReg	Train (Salary)	0.775217	0.770507	0.769901
Manual LogReg	Test (Salary)	0.719606	0.78963	0.724566
Scikit-learn LogReg	Train (Salary)	**0.7900**	0.785553	**0.7851**
Scikit-learn LogReg	Test (Salary)	0.730071	**0.8010**	0.738148
Manual MLP	Train (Salary)	0.794071	0.78766	0.786938
Manual MLP	Test (Salary)	0.733909	0.807167	0.741459
Scikit-learn MLP	Train (Salary)	**0.9741**	**0.9738**	**0.9739**
Scikit-learn MLP	Test (Salary)	0.718174	0.716648	0.717402

16.5.1 Overall Accuracy

Manual Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.7809, Test Accuracy: 0.7123
- **Salary Dataset:** Train Accuracy: 0.7708, Test Accuracy: 0.7565

Scikit-learn Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.7738, Test Accuracy: 0.7016
- **Salary Dataset:** Train Accuracy: 0.7860, Test Accuracy: 0.7705

Manual MLP Results:

- **AVC Dataset:** Train Accuracy: 0.8062, Test Accuracy: 0.7162
- **Salary Dataset:** Train Accuracy: 0.7882, Test Accuracy: 0.7725

Scikit-learn MLP Results:

- **AVC Dataset:** Train Accuracy: 0.9947, Test Accuracy: 0.8924
- **Salary Dataset:** Train Accuracy: 0.9739, Test Accuracy: 0.7925

Observations:

- **AVC Dataset:** ADASYN improved training performance but did not significantly enhance test performance. Scikit-learn MLP showed signs of overfitting.
- **Salary Dataset:** ADASYN led to balanced performance improvements, with Scikit-learn MLP showing high training accuracy but signs of overfitting.

16.6 Tomek Links

Tomek Links is an undersampling technique that identifies and removes the majority class samples that are closest to the minority class, thereby creating a more balanced dataset and potentially improving model performance.

Table 34: Comparative Table for Tomek Links - AVC Dataset

Algorithm	**Dataset**	**Precision**	**Recall**	**F1-Score**
Manual LogReg	Train (AVC)	**0.5784**	**0.5024**	**0.4943**
Manual LogReg	Test (AVC)	0.463271	0.499472	0.480691
Scikit-learn LogReg	Train (AVC)	0.478255	0.5	0.488886
Scikit-learn LogReg	Test (AVC)	0.463307	0.5	0.480955
Manual MLP	Train (AVC)	0.478255	0.5	0.488886
Manual MLP	Test (AVC)	0.463307	0.5	0.480955
Scikit-learn MLP	Train (AVC)	**0.8118**	**0.5056**	**0.5002**
Scikit-learn MLP	Test (AVC)	0.463271	0.499472	0.480691

Table 35: Comparative Table for Tomek Links - Salary Dataset

Algorithm	**Dataset**	**Precision**	**Recall**	**F1-Score**
Manual LogReg	Train (Salary)	0.787294	0.753162	0.76726
Manual LogReg	Test (Salary)	0.753141	0.737349	0.744509
Scikit-learn LogReg	Train (Salary)	**0.8069**	**0.7727**	**0.7871**
Scikit-learn LogReg	Test (Salary)	0.775146	0.756314	0.764803
Manual MLP	Train (Salary)	0.8044	0.781617	0.791766
Manual MLP	Test (Salary)	0.774383	0.765	0.769456
Scikit-learn MLP	Train (Salary)	**0.9746**	**0.9820**	**0.9782**
Scikit-learn MLP	Test (Salary)	0.728436	0.747802	0.736712

16.6.1 Overall Accuracy

Manual Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.9538, Test Accuracy: 0.9256
- **Salary Dataset:** Train Accuracy: 0.8388, Test Accuracy: 0.8275

Scikit-learn Logistic Regression Results:

- **AVC Dataset:** Train Accuracy: 0.9565, Test Accuracy: 0.9266
- **Salary Dataset:** Train Accuracy: 0.8477, Test Accuracy: 0.8325

Manual MLP Results:

- **AVC Dataset:** Train Accuracy: 0.9565, Test Accuracy: 0.9266
- **Salary Dataset:** Train Accuracy: 0.8484, Test Accuracy: 0.8330

Scikit-learn MLP Results:

- **AVC Dataset:** Train Accuracy: 0.9568, Test Accuracy: 0.9256
- **Salary Dataset:** Train Accuracy: 0.9834, Test Accuracy: 0.7975

Observations:

- **AVC Dataset:** Tomek Links slightly improved the performance without overfitting.
- **Salary Dataset:** Tomek Links led to better performance with no signs of overfitting for Logistic Regression models but some overfitting for the MLP model.

17 Final Conclusions

17.1 Overview

This section provides a summary of the performance improvements and challenges associated with each resampling technique applied to the Logistic Regression and Multi-Layer Perceptron (MLP) models on the AVC and Salary datasets. The goal is to identify the most effective approach and understand the trade-offs involved in each technique.

17.2 Overall Algorithm Comparison

Based on the analysis and comparison of Logistic Regression (LogReg) and Multi-Layer Perceptron (MLP) models across different resampling techniques, we can draw the following conclusions:

- **Logistic Regression:** The Scikit-learn LogReg model generally performed better than the manual implementation, particularly with SMOTEN and SVMSMOTE techniques. These techniques enhanced recall and F1-score significantly. However, LogReg models showed less overfitting compared to MLP models.
- **Multi-Layer Perceptron:** The Scikit-learn MLP model exhibited the highest performance metrics on the training sets across all resampling techniques, but it frequently suffered from overfitting, leading to lower test performance. Manual MLP models demonstrated more balanced performance but lower overall metrics.
- **Best Techniques:** SMOTEN and SVMSMOTE were the most effective resampling techniques, providing significant improvements in recall and F1-score for both LogReg and MLP models. These techniques were particularly beneficial for Scikit-learn LogReg models.
- **Overfitting Issues:** Overfitting was a recurrent problem in Scikit-learn MLP models, especially with techniques like SMOTE and ADASYN. Regularization and more sophisticated model tuning are necessary to mitigate this issue.
- **Model Recommendations:** For applications requiring high precision and balanced performance without significant overfitting, the Scikit-learn LogReg model with SVMSMOTE or SMOTEN is recommended. For scenarios where MLP is preferred, careful tuning and regularization are crucial to prevent overfitting.

These conclusions provide a comprehensive overview of the performance of different models and resampling techniques applied to the AVC and Salary datasets. The choice of model and technique should be guided by the specific requirements of the application, focusing on balancing precision, recall, and the risk of overfitting.

17.3 Summary of Resampling Techniques

Table 36: Summary of Resampling Techniques (Part 1)

Technique	Advantages	Disadvantages
Normal (Base Model)	• Simplicity and ease of implementation. • Baseline for comparison with other techniques.	• Struggles with class imbalance, leading to lower recall and F1-scores for the minority class. • Potential for underfitting in complex models.
SMOTE	• Significant improvement in recall and F1-score. • Effective in balancing class distribution.	• High risk of overfitting, especially in MLP models. • Synthetic samples might not represent real data accurately.
SMOTEN	• Enhanced recall and F1-score, especially for Scikit-learn LogReg. • Better handling of nominal features.	• Overfitting in MLP models. • Lower performance in manual models compared to other techniques.
SVMSMOTE	• Good balance between precision and recall. • Improved test performance for LogReg models.	• Overfitting in MLP models. • Less improvement in manual models compared to LogReg.

Table 37: Summary of Resampling Techniques (Part 2)

Technique	Advantages	Disadvantages
BORDERLINESMOTE	• Balanced improvement in recall and precision. • High performance on training sets.	• Overfitting in MLP models. • Less improvement in manual models.
ADASYN	• Enhanced performance metrics, particularly for Scikit-learn models. • Effective in generating more realistic synthetic samples.	• Overfitting in MLP models. • Lower performance in manual models.
TomekLinks Undersampler	• Improvement in precision and recall. • Effective in removing noisy samples.	• Overfitting in Scikit-learn MLP models. • Balanced but lower performance in manual models.