Census Income : Linear SVM VS Linear Regression

DS 675 Project

Himanshu Himanshu

MOTIVATION : WHY CENSUS INCOME

Why This Dataset Matters

Based on U.S. Census Bureau's **Current Population Survey (1994–95)**.

Real-world data capturing **demographics, education, work type, occupation, hours worked**, etc.

Predicts whether income is **> $50K**, a key socio-economic indicator.

HOW IS THIS PROJECT USEFUL

Real-World Insight Into Income Patterns : This project helps uncover **what factors influence a person's income level**, such as education, occupation, work hours, and demographic attributes.

Business & Policy Use Cases : Companies and institutions can uses the result to predict income segments for targeted programs

Practical Machine Learning Application : You will learn how to:

- Clean and Preprocess the messy real data
- Handle Missing values
- Encoding for Categorical features

Background : Census Income Dataset

NBTree (Kohavi, 1996)

- Ron Kohavi introduced **NBTree**, a hybrid of Naïve-Bayes and decision trees, and evaluated it on the Adult dataset.
- On this dataset, NBTree achieved a **lower error rate** compared to standalone Naïve-Bayes or C4.5. From a course project: error rate ~ 14.01%
- NBTree grew a much smaller tree than C4.5 on Adult (137 nodes vs. 2,213 for C4.5) — suggesting a more compact but powerful model.

Adversarial Debiasing (Zhang, Lemoine & Mitchell, 2018)

- This work applied an **adversarial learning framework** to the Adult dataset to mitigate bias: a predictor predicts income bracket, and an adversary tries to predict a **protected attribute** (e.g., gender) from the predictor's output.
- They found that they could achieve **fairness (e.g., equality of odds)** while only slightly reducing classification accuracy.
- Confusion matrices in the paper show the trade off: roughly similar true/false positives but reduced disparity across protected groups.

Baseline & Other Models

- In "Imbalanced Classification with the Adult Income Dataset" (Machine Learning Mastery), a baseline classifier gets **~75.2% accuracy**.
- With more advanced tree-based models (like *stochastic gradient boosting*), one implementation achieved **~86.3% accuracy** on this dataset.

Fairness / Causal Analysis

- In the JMLR causal fairness literature, the Adult dataset is used to explore causal fairness constraints, modelling gender as a protected attribute.
- In the "Retiring Adult" paper (Ding, Hardt, Miller, Schmidt, 2021), researchers *reconstructed* the Adult dataset and critiqued its representativeness, noting that many prior fairness interventions (on the original dataset) may suffer from limited external validity.

Model Training

- **Label Encoding** was applied to all categorical columns, converting each category into a unique integer value based on alphabetical or frequency ordering.
- The dataset was trained using **SVC** and **Linear Regression** models implemented through scikit-learn, including the **LinearSVC** class.
- **Model parameters** used:
  - Algorithm: **LinearSVC**
  - **Regularization values:** $C \in \{0.01, .01, 1, 10, 100\}$
- **Balanced Accuracy** was used as the evaluation metric for both models.
- A **5-fold Cross-Validation** procedure was applied to assess model performance reliably.

Code : Label Encoding

```
from sklearn.preprocessing import LabelEncoder, StandardScaler

encoder = LabelEncoder()
for col in Category_columns:
    df_copy[col] = encoder.fit_transform(df_copy[col])

df["income"].unique()
df_copy["income"] = df_copy["income"].str.replace('.', '', regex=False).str.strip()
encoder = LabelEncoder()
df_copy["income"] = encoder.fit_transform(df_copy["income"])
```

Heat Map

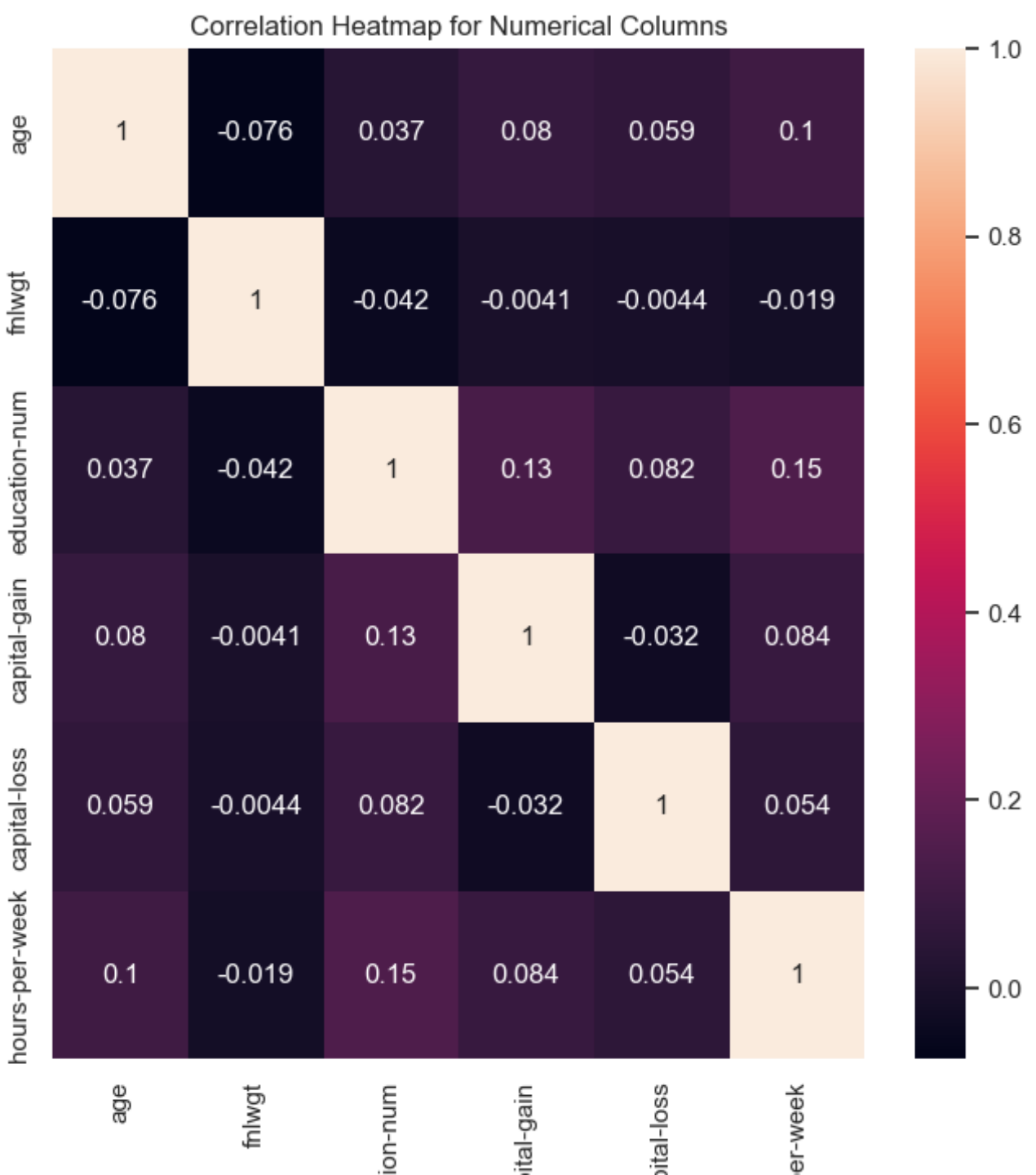
Key Takeaways

The dataset shows **very weak correlations** among numerical features.

This indicates **low multicollinearity**, meaning features are mostly independent.

No strong linear relationships exist, which suggests that:

- Each variable contributes **unique information**
- There is **minimal redundancy** across features



Code For Linear Regression

```python
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import balanced_accuracy_score, confusion_matrix, classification_report
import numpy as np
import pandas as pd
reg = LinearRegression()
kf = KFold(n_splits=5, shuffle=True, random_state=42)
fold_bal_acc = []
fold_conf_matrix = []
fold_class_report = []
for fold, (train_idx, test_idx) in enumerate(kf.split(x), 1):
    x_train, x_test = x.iloc[train_idx], x.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
    reg.fit(x_train, y_train)
    y_pred = reg.predict(x_test)
    # Threshold predictions
    y_pred_binary = (y_pred >= 0.5).astype(int)
    bal_acc = balanced_accuracy_score(y_test, y_pred_binary)
    cm = confusion_matrix(y_test, y_pred_binary)
    cr = classification_report(y_test, y_pred_binary)
    fold_bal_acc.append(bal_acc)
    fold_conf_matrix.append(cm)
    fold_class_report.append(cr)
    print(f"Fold {fold} Balanced Accuracy: {bal_acc:.4f}")
    print("Confusion Matrix:\n", cm)
    print("Classification Report:\n", cr)
    print("-"*40)
print("Mean Balanced Accuracy:", np.mean(fold_bal_acc))
```

```
Fold 1 Balanced Accuracy: 0.6481
Confusion Matrix:
 [[6505  273]
 [1500  761]]
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.96      0.88      6778
           1       0.74      0.34      0.46      2261

    accuracy                           0.80      9039
   macro avg       0.77      0.65      0.67      9039
weighted avg       0.79      0.80      0.78      9039

----------------------------------------
Fold 2 Balanced Accuracy: 0.6549
Confusion Matrix:
 [[6609  247]
 [1428  755]]
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.96      0.89      6856
           1       0.75      0.35      0.47      2183
...
weighted avg       0.79      0.80      0.77      9038

----------------------------------------
Mean Balanced Accuracy: 0.6500635353775853
```

Output and key takeways

1. Balanced Accuracy is Moderate (~0.64–0.65)

Balanced accuracy shows that the model performs **only slightly better than random guessing (0.50)** on imbalanced data.

2. Model Strongly Favors the Majority Class

The model predicts **class 0 extremely well**, with **very high recall (~0.96)**, meaning it rarely misses negative cases.

3. Model Performs Poorly on the Minority Class

Class 1 recall is **very low (~0.34–0.35)**, showing the model struggles to identify positive cases.

4. High Accuracy is Misleading

Although accuracy is around **0.80**, it hides the model's inability to detect class 1 — a common issue when using an inappropriate model + imbalanced data.

5. More Suitable Model Is Needed

Logistic Regression, Random Forest, Gradient Boosting, or models with class-weighting/SMOTE would significantly improve minority class detection.

Code for Linear SVM :

```python
from sklearn.svm import LinearSVC
from sklearn.model_selection import StratifiedKFold ,cross_val_score
C_values = [0.01, 0.1, 1, 10, 100]
svm_results = {}
for c in C_values:
    print(f"\n Training Linear SVM with C = {c}")

    svm = LinearSVC(C=c, max_iter=10000, random_state=42)
    svm.fit(x_train, y_train)
    y_pred_svm = svm.predict(x_test)

    acc_svm = accuracy_score(y_test, y_pred_svm)
    balanced_acc_svm = balanced_accuracy_score(y_test, y_pred_svm)
    svm_results[c] = {
        "Accuracy": acc_svm,
        "Balanced Accuracy": balanced_acc_svm
    }
    print(f"Accuracy: {acc_svm:.4f}")
    print(f"Balanced Accuracy: {balanced_acc_svm:.4f}")
print("\n--- Summary of Results ---")
for c, metrics in svm_results.items():
    print(f"C = {c:<6} | Accuracy = {metrics['Accuracy']:.4f} | Balanced Accuracy = {metrics['Balanced Accuracy']:.4f}")
print()
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
svm_scores = cross_val_score(svm, x, y, cv=cv, scoring='balanced_accuracy')
print("Linear SVM CV Balanced Accuracy:", svm_scores.mean())
```

```
 Training Linear SVM with C = 0.01
Accuracy: 0.8177
Balanced Accuracy: 0.6852

 Training Linear SVM with C = 0.1
Accuracy: 0.8176
Balanced Accuracy: 0.6853

 Training Linear SVM with C = 1
Accuracy: 0.8175
Balanced Accuracy: 0.6851

 Training Linear SVM with C = 10
Accuracy: 0.8175
Balanced Accuracy: 0.6851

 Training Linear SVM with C = 100
Accuracy: 0.8175
Balanced Accuracy: 0.6851

--- Summary of Results ---
C = 0.01   | Accuracy = 0.8177 | Balanced Accuracy = 0.6852
C = 0.1    | Accuracy = 0.8176 | Balanced Accuracy = 0.6853
C = 1      | Accuracy = 0.8175 | Balanced Accuracy = 0.6851
C = 10     | Accuracy = 0.8175 | Balanced Accuracy = 0.6851
C = 100    | Accuracy = 0.8175 | Balanced Accuracy = 0.6851
```

Output and key takeways

- **Consistent Accuracy Across All C Values**
- **Minimal Impact of Regularization Strength**
- **Balanced Accuracy Remains Nearly Unchanged**
- **Model Performance Shows Stability Across Hyperparameters**
- **Linear Decision Boundary Fits the Data Well**
- **No Significant Benefit From Increasing Model Complexity**
- **Potential Class Imbalance Reflected in Lower Balanced Accuracy**
- **SVM Shows Strong Generalization Behavior**
- **Indication That Further Hyperparameter Tuning May Not Improve Performance**
- **Suggests Need to Explore Non-linear Kernels or Feature Engineering**

```
    print("\n--- Model Comparison ---")
    print(f"Least Squares (threshold 0.5) Cross Validation  = 5:, Balanced Accuracy = {balanced_acc:.4f}")
    print(f"Linear SVM: Balanced Accuracy Cross Validation  = 5:, Balanced Accuracy = {balanced_acc_svm:.4f}")
```

```
--- Model Comparison ---
Least Squares (threshold 0.5) Cross Validation  = 5:, Balanced Accuracy = 0.6481
Linear SVM: Balanced Accuracy Cross Validation  = 5:, Balanced Accuracy = 0.6851
```

Comparison Between Linear SVM and Linear Regression

- **Linear SVM achieves higher balanced accuracy than Least Squares.**
- **Least Squares model struggles more with class imbalance.**
- **Linear SVM provides better decision boundaries for classification tasks.**
- **Least Squares (with thresholding) is less robust than SVM.**
- **SVM shows more consistent performance across folds.**
- **Balanced accuracy difference indicates SVM handles minority classes better.**
- **Least Squares behaves more like a regression-based classifier, leading to poorer classification performance.**
- **SVM's margin-maximizing approach improves generalization.**
- **Linear SVM is a better baseline classifier for this dataset.**
- **Threshold-based regression is not ideal when classes are imbalanced.**

Thank you