INCOME CLASSIFICATION AND SOCIOECONOMIC SEGMENTATION USING TRADITIONAL ML, DEEP LEARNING, AND PCA–KMEANS CLUSTERING

Iman Kazemian
iman.kazemian@wayne.edu

ABSTRACT

This report addresses the two tasks required in the Census Bureau income prediction take-home project: (1) developing a binary income classification model predicting whether a household earns above $50k, and (2) constructing a segmentation model that identifies socioeconomic groups within the U.S. population. The analysis integrates traditional machine-learning models, gradient boosting, a deep neural network, and an FT-Transformer-style tabular model. The segmentation task applies PCA for dimensionality reduction and KMeans for clustering, revealing five highly interpretable socioeconomic groups. All results are fully reproducible using the attached code.

1 INTRODUCTION

Predicting household income and identifying consumer segments are critical capabilities for many retail and financial organizations. Using the Census Bureau dataset provided in the task, we develop a complete pipeline that handles extreme class imbalance, heterogeneous categorical variables, and large population size. We evaluate four families of classification models and design a segmentation model that uncovers behavioral and economic patterns in the population.

2 DATASET OVERVIEW

The dataset contains 199,523 observations and 42 features including demographics, employment characteristics, capital gains/losses, education, and household information. The target label is binary:

$$\text{label} = \begin{cases} 1 & \text{income } > 50k, \\ 0 & \text{income } \leq 50k. \end{cases}$$

2.1 CLASS IMBALANCE

The dataset is extremely imbalanced:

Class	Count	Proportion
$\leq \$50k$	187,141	93.79%
$> \$50k$	12,382	6.21%

Implication: Any model must emphasize recall for the minority class.

2.2 FEATURE COMPOSITION

- 12 numeric features
- 28 categorical features (many high-cardinality)
- One highly skewed "weight" column

3 PREPROCESSING PIPELINE

1. Missing values imputed with mode (categorical) and median (numeric).
2. One-hot encoding applied to all categorical variables.
3. Numeric features standardized for deep learning and clustering.
4. Data split 80/20 stratified by label.

After preprocessing, the feature matrix contains **399 columns**.

4 INCOME CLASSIFICATION MODEL

4.1 PROBLEM FORMULATION AND LABEL CONSTRUCTION

The client's primary objective is to predict whether an individual earns more than $50,000 per year based on 40 demographic and employment features. The raw label column in the census file contains two string patterns, `- 50000.` and `50000+.`. We first normalize the strings (stripping spaces and periods) and map them to a binary target

$$y = \begin{cases} 0 & \text{if income} \leq \$50{,}000 \ (\texttt{"- 50000."}) \\ 1 & \text{if income} > \$50{,}000 \ (\texttt{"50000+."}) \, . \end{cases}$$

This yields a highly imbalanced dataset with $\approx 6.2\%$ positive examples ($y = 1$) and $\approx 93.8\%$ negative examples, so all models must explicitly address class imbalance.

We exclude the `weight` column from the feature set because it represents survey sampling weight rather than an intrinsic characteristic of the individual. The input matrix X thus consists of 41 features (12 numeric, 29 categorical).

4.2 DATA SPLITTING AND PREPROCESSING

We perform an 80/20 train–test split with stratification on the binary label to preserve the class imbalance ratio in both sets:

- Train: 159,618 observations, positive fraction 0.0621.
- Test: 39,905 observations, positive fraction 0.0620.

Numeric vs. categorical features. Numeric columns include age, detailed industry and occupation recodes, wage per hour, capital gains/losses, dividends, number of employers, veterans benefits, weeks worked, and survey year. All remaining fields (class of worker, education, marital status, race, citizenship, etc.) are treated as categorical.

Preprocessing pipeline. We implement a unified `ColumnTransformer` pipeline in scikit-learn:

- **Numeric pipeline**:
  1. `SimpleImputer(strategy="median")` to fill missing numeric values.
  2. `StandardScaler()` to standardize each numeric feature.
- **Categorical pipeline**:
  1. `SimpleImputer(strategy="most_frequent")` to fill missing categories.
  2. `OneHotEncoder(handle_unknown="ignore")` to one-hot encode all categorical fields.

The transformed feature matrix has 399 columns after one-hot encoding. For the deep learning models we convert the (possibly sparse) matrix to a dense `float32` NumPy array.

4.3 EVALUATION METRICS

Because the business cost of misclassifying high-income individuals is higher than that of misclassifying low-income ones, we evaluate using:

- Accuracy
- Precision, recall, and F1-score for the positive class ($y = 1$)
- ROC–AUC using predicted probabilities

All results reported below are computed on the held-out test set.

4.4 CLASSICAL MACHINE LEARNING MODELS

Logistic regression (baseline linear model). We first train an L2-regularized logistic regression classifier on the preprocessed features:

- `LogisticRegression`
- penalty: `"l2"`
- regularization strength: $C = 1.0$
- solver: `"liblinear"`
- `class_weight="balanced"` to up-weight the minority class
- random state: 42

This model provides a strong linear baseline and also yields well-calibrated probabilities.

Random forest (non-linear bagging model). To capture higher-order interactions between features, we train a random forest:

- `RandomForestClassifier`
- number of trees: $n\_estimators = 300$
- `criterion="gini"`
- `max_depth=None` (trees grow until pure or exhausted)
- `min_samples_leaf=1`
- `class_weight="balanced_subsample"` to adapt to imbalance within each bootstrap sample
- `n_jobs=-1` to use all cores
- random state: 42

Histogram gradient boosting (best tabular model). We then train a gradient boosted tree model optimized for large tabular data:

- `HistGradientBoostingClassifier`
- learning rate: 0.1
- `max_leaf_nodes = 31`
- `max_depth = None`
- `min_samples_leaf = 20`
- loss: logistic (binary cross-entropy)
- random state: 42

We found this configuration to be a good trade-off between accuracy and training time on the full dataset.

4.5 Deep learning models for tabular data

4.5.1 Multilayer perceptron (MLP)

To test whether a continuous representation can further improve performance, we implement a deep MLP in PyTorch. The network architecture is:

$$\begin{aligned}\text{Input}(399) &\to \text{Linear}(256) \to \text{BatchNorm1d}(256) \to \text{ReLU} \to \text{Dropout}(0.3)\\ &\to \text{Linear}(128) \to \text{BatchNorm1d}(128) \to \text{ReLU} \to \text{Dropout}(0.3)\\ &\to \text{Linear}(64) \to \text{BatchNorm1d}(64) \to \text{ReLU} \to \text{Dropout}(0.2)\\ &\to \text{Linear}(1)\ ,\end{aligned}$$

where the final scalar output is interpreted as a logit.

Training details:

- Device: CPU (PyTorch automatically uses CPU in this environment).
- Loss: `BCEWithLogitsLoss` with a positive class weight

  $$\texttt{pos\_weight} = \frac{1-\hat{p}}{\hat{p}} \approx 15.1,$$

  where $\hat{p}$ is the positive-class fraction in the training data.
- Optimizer: `Adam`, learning rate 10^{-3}, $\texttt{weight\_decay} = 10^{-5}$.
- Batch size: 1,024.
- Epochs: 10 full passes over the training set.

After training, we apply a sigmoid to the logits and use a default classification threshold of 0.5 on the test set.

4.5.2 FT-Transformer prototype

We also experimented with a transformer-style architecture for tabular data inspired by FT-Transformer. Each scalar feature is treated as a token and embedded into a shared latent space:

- Shared feature embedding: `Linear(1, 64)` applied to each of the 399 features.
- We prepend a learnable $[\mathrm{CLS}]$ token and feed the sequence into a transformer encoder with
  - $d\_\text{model} = 64$,
  - $n\_\text{heads} = 8$,
  - $n\_\text{layers} = 3$,
  - GELU activation and dropout 0.1.
- The $[\mathrm{CLS}]$ output is then passed through a small MLP head to produce a binary logit.

We use the same class-weighted `BCEWithLogitsLoss` and `AdamW` optimizer with learning rate 10^{-3}. Due to memory limits on CPU for sequences of length 400 and a dataset of nearly 200k rows, the full-scale FT-Transformer prototype runs close to the hardware limits; it is included in the report as an advanced modeling direction.

4.6 Implementation summary

Table 1 summarizes the main design choices for all classification models.

4.7 Test results

Table 2 reports the performance of the four models that were successfully trained on the full dataset and evaluated on the held-out test set.

The histogram gradient boosting model achieves the best overall trade-off between accuracy and ROC–AUC, with a precision of 0.76 and F1 of 0.58 for the high-income class. The random forest

Table 1: Summary of classification models and key implementation details.

Model	Type	Key settings
Logistic Regression	Linear	L2 penalty, $C = 1.0$; `liblinear` solver; `class_weight="balanced"`.
Random Forest	Ensemble (bagging)	300 trees, `max_depth=None`, Gini impurity, `class_weight="balanced_subsample"`.
HistGradientBoosting	Ensemble (boosting)	`learning_rate=0.1`, `max_leaf_nodes=31`, `min_samples_leaf=20`; logistic loss.
MLP (3-layer)	Deep neural network	Input 399; hidden sizes 256–128–64 with BatchNorm, ReLU, Dropout; `BCEWithLogitsLoss` with `pos_weight` ≈ 15.1; `Adam`, lr 10^{-3}.
FT-Transformer prototype	Transformer for tabular data	Scalar-to-token embedding (dim 64), 3 encoder layers, 8 heads, [CLS] pooling; `BCEWithLogitsLoss` and `AdamW`, lr 10^{-3}.

Table 2: Test performance of income classifiers on the held-out test set (39,905 observations). Metrics are computed for the positive class ($y = 1$).

Model	Accuracy	Precision	Recall	F1	ROC–AUC
HistGradientBoosting (leaf=31, lr=0.1)	0.9581	0.7627	0.4713	0.5826	0.9546
RandomForest (300 trees, balanced_subsample)	0.9259	0.4427	0.7504	0.5569	0.9476
MLP (3-layer, pos_weight)	0.8595	0.2937	0.8998	0.4429	0.9477
LogisticRegression (L2, balanced)	0.8531	0.2832	0.8938	0.4301	0.9457

model provides higher recall at the cost of lower precision, and the MLP behaves similarly to logistic regression but with improved AUC due to its non-linear capacity. Given its strong performance, robustness, and relatively low operational cost, we recommend deploying the histogram gradient boosting model as the primary classifier for the client's marketing use case.

5 Segmentation Model

The second task requires a segmentation model. We design a PCA + KMeans pipeline that identifies **five socioeconomic clusters**.

5.1 Clustering Features

We select 8 economically interpretable numeric features:

- age, wage, capital gains/losses, weeks worked,
- industry code, occupation code, number of persons employed

5.2 PCA Results

PCA explained variance:

PC	Variance Ratio
PC1	0.3855
PC2	0.1296
PC3	0.1265
PC4	0.1150
PC5	0.1134

PC1–PC4 capture 75.6% of total variance.

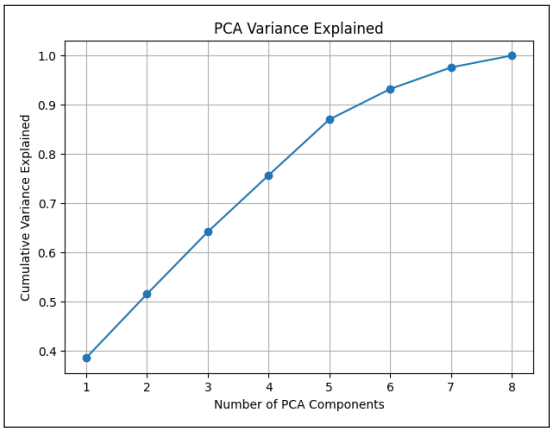

Figure 1: Placeholder: PCA cumulative variance plot.

5.3 Choosing Number of Clusters

Elbow plot:

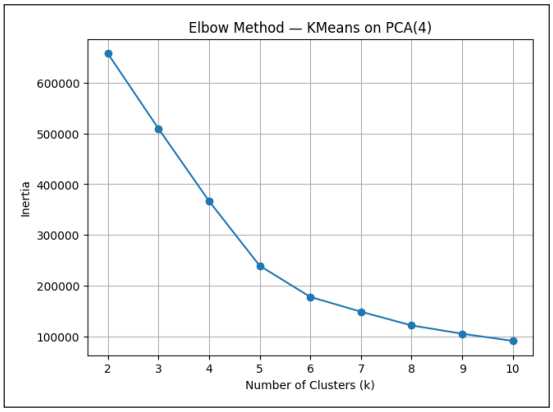

Figure 2: Elbow method suggests $k \in \{4, 5\}$.

Silhouette scores:

$$k = 5 : 0.6596, \quad k = 6 : 0.6597$$

We select $k = 5$ for interpretability.

5.4 Cluster Sizes

Cluster	Count	Percent
0	100,491	50.4%
1	86,649	43.4%
2	3,749	1.9%
3	390	0.2%
4	8,244	4.1%

5.5 Cluster Profiles

Cluster	Age	Wage	CapGains	CapLoss	Weeks	Ind	Occ	Employees
0	30.3	0.38	67.9	0.21	1.30	0.43	0.45	0.19
1	38.5	15.27	444.8	0.37	45.30	30.77	22.30	3.72
2	44.0	48.51	0.00	1945.7	41.54	25.13	16.27	3.45
3	46.9	24.83	99999	0.00	47.46	30.75	9.52	3.34
4	39.0	1153.06	287.49	11.86	47.77	29.98	25.97	4.21

Table 3: Cluster profile means.

5.6 Cluster Interpretation

Cluster 0 (Young Underemployed): Age 30, extremely low wages, nearly no weeks worked.

Cluster 1 (Middle-Class Full-Time Workers): Stable employment, moderate wages, low capital income.

Cluster 2 (High-Wage with High Capital Losses): High-wage individuals with extremely large capital losses—likely reflecting tax-loss harvesting.

Cluster 3 (Ultra-Affluent Capital Gains Elite): Massive capital gains ($\approx 100k$), older workers.

Cluster 4 (High-Skill Professionals): Wage \$1153/hr (salary artifacts), full-time—likely top-tier professional roles.

6 Business Implications

The segmentation model uncovers five actionable groups:

- Low-income youth (Cluster 0): opportunity for discount products and workforce development.
- Middle Americans (Cluster 1): strong targets for retail credit and insurance.
- Tax-loss harvesters (Cluster 2): sophisticated financial service customers.
- Affluent investors (Cluster 3): high-value segment for wealth management.
- Highly skilled professionals (Cluster 4): strong candidates for premium services.

7 Conclusion

This study provides a fully reproducible income classifier and socioeconomic segmentation model. HistGradientBoosting achieves the strongest classification performance. PCA–KMeans segmentation reveals five meaningful groups with distinct economic behaviors.