An Analysis of the 1994 United States Census

Brent Caldwell, Olivia Ryan, Teddy Amare

**Introduction and Description**

The following report contains an analysis of the 1994 United States Census. This dataset was originally extracted from the US Census website (<http://www.census.gov/ftp/pub/DES/www/welcome.html>), but was downloaded by us from (<https://archive.ics.uci.edu/ml/datasets/Census-Income+%28KDD%29>). The tasks we performed for this section are data cleaning, classification, and clustering. In the original dataset, each person is represented by their age, work-class, estimated representative weight, education, a numeric education value, marital status, occupation, relationship, race, sex, capital gains, capital losses, hours worked per week, native country, and an estimated yearly income total classified by over or under 50,000 USD over the previous year.

**Data Cleaning**

When examining the data for classification and clustering tasks, we found that the feature ‘education’ contained the same values as the ‘education-numeric’ feature, so to prevent redundancies, the former was dropped.

There were also three characteristics that contained missing values: work-class, occupation, and native country. Work class and occupation are similar columns, in that they both are based on the types of jobs each member of the population had over the 1994 year. An analysis of the data showed that the context of the missing values in these two characteristics was that those people were unemployed, so their data was omitted. To remedy the missing values, their description was changed to ‘unemployed’. The ‘native country’ unfortunately was more difficult to pinpoint an exact reason for the missing data, so it was settled to create a new value titled ‘other’.

**Clustering**

The Clustering alogirthm that was performed is called K-Means. This clustering approach is an unsupervised machine learning algorithm to partition our data set into k clusters based on their similarity. To obtain the optimal k, we have implemented two algorithms: elbow method and silhouette scoer.

The elbow method is completed by plotting Within-Cluster sum of squres against the number of clusters. We find the “elbow” point by finding the dip where the graph changes it’s shape.

![Chart, line chart

Description automatically generated]()

The silhouette method finds the optimal k by calculating the mean silhouette coefficient for different values of k. The higher the score, the better the clustering, meaning the points are matched to their assigned cluster. Before completing the clustering, we dropped the education attribute because there was another attribute that had the same numerical values instead of the categorical one.

Using both methods, we find the optimal k to be 3 which was consistent across multiple iterations.

k=2, the clusters are primarily separated by Age, where one of the clusters appears for younger individuals vs older ones. We also see overlap between Education and hours per week between the two clusters.

k=3, we separate the clusters by Age and Education and we can see more into younger population with higher level of education and the opposite representing the higher level. Some attributes don’t really have a big effect on the clustering. Relationship attribute can be dropped for example in our case.

The clustering has provided insights to the dataset that we wouldn’t normally see. Clusters have showed us different population ranging from a certain age with the relationship they have with other attributes.

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

**Research Question**

The goal of this dataset is to be able to classify citizens of the United States into one of two income brackets: Greater than $50,000 net annual income, and less than or equal to $50,000 annual income. To answer this question, the following methods were used: rule-based classification, Naïve Bayes classification, and artificial neural networks.

**Classification Methodologies**

Rule-Based Classification

The next method used was the Wittgenstein package version of RIPPER to generate association rules. The dataset was split in a 40/60 ratio for training/testing, and then trained to detect rules that would match ‘census\_income is less than or equal to $50,000’. The method then printed out all the association rules the training data matched with the greatest correlation to a person in that income bracket. Once the rules were generated, the testing split was examined to determine precision and recall potential of the rules.

During the testing and training, the precision of correctly identifying a person as less than or equal to $50,000 was calculated to be 94.21%. However, the recall was calculated to be 70.18%. This means that the rules matched to a severe majority of the persons in the correct income bracket, but that the rules also tended to match with people who had an income of greater than $50,000, creating many false positives.

A second test was run on the dataset to see if rule generation would have greater success matching rules to people in the upper income bracket. The precision of this test was 94.09%, and the recall was 70.12%

**Naive Bayes Classification**

Naive Bayes is a classification algorithm that is based on Bayes' theorem, which states that the probability of a hypothesis (data point belonging to a particular class for instance) is proportional to the likelihood of the evidence (the features of the data point) given that hypothesis, multiplied by the prior probability of the hypothesis. The “naice” in Naive Bayes is so called because it makes a simplifying assumption that the features are conditionally independent given the class label, even though they may be correlated.

The naive\_bayes function provided performs Naive Bayes classification on the given dataset. We perform one-hot encode the categorical vairables for classification purpose and drop the original categorical values. The class label is also converted to a binary integer representation where 0 represents the class <=50K and 1 represents the class >50K.

The function then splits the dataset into training and test sets using the train\_test\_split function from the sklearn library. It fits a Gaussian Naive Bayes model to the training data using the GaussianNB class from the sklearn.naive\_bayes module. The accuracy of the model is then printed to the console using the score method on the test data. The predicted class labels are also computed using the predict method and stored in a DataFrame along with the actual class labels.

We’ve created a confusion matrix heatmap showing the number of true positives, false positives, true negatives, and false negatives for the classifier.

Finally, the function prints the first 7 actual class labels, predicted class labels, and probability of the predicted class labels for the test data. It also prints the cross-validation score for the classifier using the cross\_val\_score function from the sklearn.model\_selection module with 3 folds.

We’ve found the model accuracy to be 79%. This indicated that the naive bayes classifier predicted the class labels for 79% of the test data sets. The confusion matrix and heatmap show that there were a relatively large number of True negatives, meaning the classifier correctly predicted that a data point belonged to the class label. The cross-validation score shows that the model's performance was consistent across the 3 folds.

**Outlier Detection**

This section of the project compared two anomaly detection techniques: DBSCAN and Isolation Forest. Although both methods aim to detect anomalies, they differ in their approaches to achieve this goal. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based algorithm that identifies clusters of points based on the density of nearby points. Any point that is too far from any cluster is considered an outlier. In contrast, Isolation Forest is a tree-based algorithm that isolates anomalies by randomly partitioning data points into subsets. An anomaly score is assigned to each data point based on the average path length required to isolate it in the constructed trees.

DBSCAN can be computationally expensive as it requires calculating the distances between all pairs of points within the dataset. This can make it difficult to scale to large datasets. Isolation Forest is more computationally efficient as it only requires building a set of random trees to detect anomalies. Isolation Forest is better for handling high dimensional data as compared to DBSCAN.

DBSCAN has two key parameters that require tuning: eps (maximum distance between points in a cluster) and min\_samples (minimum number of points required to form a cluster). In our project the eps was set to 2000 and min\_samples to 400 for optimal performance. Isolation Forest has fewer parameters to tune, making it easier to use.

DBSCAN detected 2660 outliers out of a total of 32,561 values, resulting in a percentage of outliers detected of 8.17%. On the other hand, Isolation Forest detected 3,249 outliers out of a total of 32,561 values, resulting in a percentage of outliers detected of 9.98%. Based on these results, we can see that Isolation Forest detected more outliers than DBSCAN. This indicates that Isolation Forest is more aggressive in identifying outliers and may be prone to some false positives.

In conclusion, both DBSCAN and Isolation Forest are useful techniques for anomaly detection that vary in their approach. Based off the findings of this project, if the goal is to identify a small number of outliers that are more accurate (low false positive rate), then DBSCAN is more suitable. However, if the goal is to identify a larger number of outliers even at the cost of some false positives, then Isolation Forest may be more suitable. There is no right or wrong choice between which technique to use for outlier detection, as both have their own strengths and weaknesses. However, the choice of these two techniques depends on the specific characteristics of the dataset and the application requirements.