# Data Mining

Assignment 1: Access control

Mohammed Al-Ogaili - 20193575

2022 - 2023

## 1 Introduction

In this report we will be going over the different steps we went through to complete the given assignment.

We were given a dataset of existing customers along with their income class (whether they earn more than 50K), and another dataset of potential customers that are not classified. So, the goal of this assignment is to correctly classify the income bracket of each customer and decide whether sending them a promotional package would be a good investment.

In a nutshell, the assignment is about trying to classify a given incomplete dataset based on existing data.

## 2 Initial analysis

The first step to building a good classifier is getting acquainted with the available data. This is exactly what we did. We first read through the different fields and their definitions and then began the analysis by loading the data into python. Now that we had the data loaded in as a DataFrame object, we could easily poke around and see how everything looks.

So, we printed out some information about the dataset, namely the unique values for each column and how many times they occurred. While it was interesting to see, it didn't provide any insight, but it did give us an idea of the kind of encoding to use for them. It was also here that we learned that the data contains many empty entries, which we will have to deal with one way or the other.

Looking the unique values for the 'class' column did help us understand the dataset much better. We got the following output:

| class | count |
|---|---|
| $\leq$ 50K | 24720 |
| $>$ 50K | 7841 |

Table 1: Count of each class in the training set

This indicates that the training dataset is highly imbalanced, so we will most likely need to do some form of under- or oversampling to make up the difference.

With these observations we felt confident enough to start with the preprocessing step.

## 3 Preprocessing the data

First thing we did in terms of preprocessing after loading in the data was removing all rows missing values. We opted for this rather than trying to populate these entries with the means of each column or any other method. The reason behind this was simplicity and because we had feared that the classifier might learn things that are not true for the given data. We do not know the exact reason behind why these rows have missing values. It might be completely random, or these rows might be somehow related. Filling in these values might make the classifier learn relations that aren't there or break relations that are present. So, we picked the safest option.

After removing these rows we removed some columns that could potentially introduce bias. The columns we manually removed are 'age', 'race', 'sex', 'native-country', and 'relationship'. We removed these columns because they could result in discriminatory and unwanted results based on someone's age, gender, race, etc.

Then we used one-hot encoding to transform the categorical columns into numerical (binary) features. We created a new column for each category/feature and populated it with a boolean indicating whether a feature is present. We used this encoding method to prevent any ordering or magnitude assumptions that could skew our classifier's predictions.

Next, we split the data into training and test sets with their respective labels. To deal with the imbalance of the dataset we decided to apply undersampling on the majority class ($< 50$K). We kept only a third of the entries.

# 4 Training the classifiers

For this assignment we decided to use a variety of classifiers provided by the scikit-learn library. We tried many classifiers and came up with the following list to compare to each other: k-nearest neighbors classifier, decision tree classifier, complement Naïve Bayes classifier, categorical Naïve Bayes classifier, Random Forest classifier, and a logistic regression classifier.

To train the classifiers we took a somewhat unusual approach. At first, we tried the usual approach of cross validating the classifier and performing a grid search to tune the hyperparameters. However, we found that doing it that way took a very long time with a not very significant improvement in the end result (profit gain on the test set). So, we decided to forego these steps and start by doing feature selection straight away. To do that, we implemented the backwards feature selection algorithm, removing features till we couldn't observe any improvement in our scoring metric (profit) anymore.

Our scoring metric is as follows: $score = \text{TP} * (980 * 0.1 - 10) - \text{FP} * (310 * 0.05 + 10)$. This metric essentially results in the expected profit gain if we were to send promotional packages to every customer classified as earning more than 50K. The idea behind scoring our classifiers by this metric is that if we were to optimize the classifiers to the maximal expected profit in the test set then they would also do the same on the potential customer data.

After finding the optimal feature set, we fit the model on the data using only these features. Then we start optimizing the prediction threshold to maximize our profits even more. Using the method '`predict_proba`' we can obtain the degree of certainty of each prediction (as positive). Using that we can look for the optimal threshold where our expected profit is the highest. Doing it this way will result in slightly more false positive predictions (precision) but our recall will improve which is the more important metric in this case. The penalty for a false positive prediction is much lower than missing a true positive prediction (losing out on potential €88 gain vs just losing €25.5).

This pretty much concludes the extent of our training procedure. It is fairly simple and optimized for run and training time rather than getting every euro of expected profit. This allowed us to test many classifiers rapidly and allowed us to experiment with the other preprocessing techniques (e.g. normalization, sampling, etc.) and hyperparameters.

# 5 Classifying the potential customers

Classifying the potential customers with each classifier and comparing them was fairly easy. All we had to do is call '`predict_proba`' and use the optimal threshold we obtained earlier. Then we could get the ids of each positively predicted row and output them to a file.

To compare the classifiers we calculated an estimate of our expected profit. Our estimate is calculated by using the precision score to get a rough estimate of how many of the predictions are correct. The full equation is:

$$estimated\_profit = \lfloor \text{precision} * \text{positives} \rfloor * (980 * 0.1 - 10) - (\text{positives} - \lfloor \text{precision} * \text{positives} \rfloor) * (310 * 0.05 + 10)$$

From our testing we estimate this approximation to be within about 5% of the actual expected profit.

Finally, when we applied everything mentioned here we got the following estimates:

```
KNN: Estimated profit: €213265.00
Decision trees: Estimated profit: €232197.50
Complement Naïve bayes: Estimated profit: €55752.50
Naïve bayes: Estimated profit: €231282.00
Random Forest: Estimated profit: €223493.00
Logistic Regression: Estimated profit: €231651.50
```

We could immediately see that the decision trees classifier, logistic regression classifier and categorical Naïve Bayes resulted in the highest estimates. Furthermore, we also observed that the logistic regression classifier tends to over-estimate, and that the Naïve Bayes classifier seemed to be the most stable with accurate estimates. So, taking all that into consideration we decided to go with the list of ids from the Naïve Bayes classifier.

# 6 Conclusion

We found this assignment to be very interesting. It was not as straight forward as one might think, and there are many ways to 'solving' this problem. It's just a matter of how much effort and time one is willing to devote to it. This was a unique and fun challenge which taught us many techniques that can be used in the real world.

# 7 Extra's

Github repo: https://github.com/Mohatje/DataMining-Classification

Promotion rowids: output/selected-customers.txt