Data Mining

Assignment 1: Classification

Noah Daniëls (s0192421)

April 2023

Introduction

In this report I discuss my solution and approach to the first assignment of the data mining course (2023) at the University of Antwerp.

The assignment was centered around classification and consisted of predicting whether potential customers have a high or low income based on demographical information. We received a labeled dataset of existing customers and an unlabeled dataset of potential clients.

Based on our predictions and some given business rules, we had to create a selection of promising customers to mail a promotion to and estimate the expected gain in revenue from this selection.

In the first section I describe my general approach for solving this problem: What tools I used, and how I evaluated and tuned the different models. The second section covers the different models I tested and the specific preprocessing steps and hyperparameters required for each. In the final section I describe how I made my final selection of potential customers and how I obtained a revenue estimation.

1 Approach

I used *sklearn* to solve this assignment. The code is available on github in the form of a jupyter notebook.

I started by loading the datasets using pandas. I inspected the data to get a feel for what was in the different columns and how many missing values there were. Next, I performed some preliminary preprocessing. I extracted the label column from the dataset and dropped the race and sex columns because I deemed it ethically questionable to use those features. I also split the data into a training and testing set using a 70-30 split. All the models described below were used with these basic preprocessing steps applied.

I made sure to base all decision such as hyperparameter tuning and model selection on the score obtained from cross validation on the training set. The test set was only used at the very end to evaluate the final chosen model and obtain the accuracy. By avoiding data leakage like this, I can be confident that the final reported accuracy is not overestimated and the chosen model is not overfit on the test data.

To achieve this, I made use of sklearn pipelines. Using pipelines, it is easy to define the preprocessing steps and the model and then perform cross validation on the entire pipeline. Sklearn also makes it easy to perform grid search for hyperparameter tuning using cross valdiation. And because everything is done with pipelines, I could even tune the parameters of the preprocessing steps.

1.1 Missing values

Because both the labeled and unlabeled data contained missing values, I could not simply drop the affected rows. Similarly, the features that had missing values seemed as if they would provide useful information so I also couldn't drop them entirely either.

Instead, I used a preprocessing imputation step in sklearn which can fill in missing values. I tried three approaches and found that they all performed similar. The first two approaches simply filly in the missing values with the mean or median for the given feature. The third approach finds the k nearest neighbours based on the columns that are not missing and computes an average among them for the missing value.

What method I ended up using depends on the model, but usually the differences were very slight and simply using the average for all models would not have made too much of a difference.

2 Models

I tested and tuned four classifier algorithms before choosing one to solve this assignment.

2.1 Naive Bayes

I used sklearn's categorical NB classifier because most of the dataset consisted of categorical data and the few numerical features could easily be discretized in bins. Specifically, I used the following categorical features: 'workclass', 'education', 'marital-status', 'occupation', and 'native-country'. And I binned 'age' into 5 equi-depth bins and 'hours-per-week' into 3 equi-width bins.

Since sklearn only works with numerical matrices, I used an ordinal encoder for the categorical features. This encoder simply assigns a unique value to each possible category within each feature.

The categorical NB classifier only has one hyper parameter which is the laplace smoothing factor. I used cross validation as described above to find the optimal value for this parameter which resulted in a smoothing factor of $\alpha = 0.5$. The final cross validation accuracy was 82.6%.

Note. Even though the assumptions for NB don't hold for our data (occupation and education are clearly not class-independent and neither are age and marital-status), we still obtain relatively high accuracy. This is because as stated during the lectures, the assumptions aren't strictly necessary in practice to obtain decent results. The catch is that the predicted probability won't be very accurate. This is a problem for us as we'll see in the last section were we need these probabilities to be accurate for our selection and revenue estimation.

2.2 k-Nearest Neighbors

The next model I tested was a KNN classifier. Like Naive Bayes, this model also works with categorical data so I reused the preprocessing from NB. The two main hyperparameters that sklearn provides are the number of neighbours to consider and whether they are weighted.

Again using grid search with cross validation as described above, I found that 20 neighbours and no weighting achieved the best performance. The final cross validation accuracy was 81.6%.

2.3 Decision Trees

The decision tree implementation of sklearn only supports ordinal numeric input. This is annoying as most of our data is categorical. We cannot simply use the oridinal encoder because the decision tree will interpret the order as having meaning. In the case of 'education-num' this is what we want, but for the other categories where there is no apparent order, we need to use one-hot encoding instead.

Specifically, I used 'workclass', 'marital-status', 'occupation', and 'native-country' as categorical data encoded this was. And I used 'age', 'hours-per-week', 'education-num', 'capital-gain', and 'capital-loss' as is.

The main hyperparameters to tune for decision trees decide how to stop growing the tree and avoid overfitting. The best combination I found with cross validation was capping the maximum number of leaf nodes at 150 and requiring at least 10 samples at each leaf. This resulted in a final accuracy of 85.9%.

I also read that decision trees can have trouble when the samples are not evenly distributed. In the labeled data, the lower income customers are over represented (75% are lower income). To remedy this, I tried randomly selecting an equal amount of samples for each class and training on this data. However, this resulted in slightly worse performance. I think this is due to the fact that the data is not that imbalanced and the decrease in training data more detrimental than the imbalance.

2.4 Random Forests

The final method I tested was random forest, which followed the same preprocessing steps as decision trees. The main parameters were again controlling when to stop growing the tree, but now also how many trees to use and how many samples or features each tree can see.

I obtained the best results by limiting the individual trees the same way as regular decision trees: by setting a maximum number of leaves. Using 100 trees in the forest which all see the same features and samples gave the best results. The final cross validation accuracy was only slightly better than regular decision trees at 86.6%.

3 Final Results

From these experiments, I selected the Random Forest model. When trained on all the training data, it achieved a test set accuracy of 86.5%.

Since the training-testing split is only needed for evaluating the model, I retrained the model on all the labeled data (training + test data) for the final selection.

3.1 Final Selection

With an accurate classifier available, all that remains is to make a selection of potential customers to send promotional material to. From the rules in the assignment, we can find a formula for the expected revenue for a potential client whose probability of having a high income is p_i:

$$\mathbb{E}[R(i)] = 0.1 \cdot 980 \cdot p_i - 0.05 \cdot 310 \cdot (1 - p_i) - 10 = 113.5 \cdot p_i - 25.5$$

Which allows us to find the minimum probability for this expected revenue to be positive:

$$\mathbb{E}[R(i)] \geq 0 \implies 113.5 \cdot p_i - 25.5 \geq 0 \implies p_i \geq 0.225$$

Thus by selecting all potential customers with an estimated probability exceeding 22.5%, we will maximize the expected revenue. Our trained decision tree classifier also has the ability to return probabilities, so this becomes a simple matter of applying our model to all potential customers and dropping those whose probability is too low.

Note that this means that we will select people who we predicted would be low income. And indeed, since our accuracy is relatively high, we will indeed send it too many people with low income. However, since the payoff for being high income is higher than the loss for low income, it remains advantageous to take this risk on the basis that we will include a couple more high income clients which we would have otherwise missed. (There is also a snippet in the notebook comparing this with the greedy approach of only selecting customers with predicted high income, which performs worse.)

This results in a list of 5837 potential clients which can be found in `selection.txt`. We can also use the probabilities to calculate the expected revenue for each client. When summing all these value we get an estimated total revenue of €226,679.