Prediction of Earnings Based on Demographic and Employment Data

I. Introduction

Background

The U.S. Department of Commerce launched Census Bureau to gather data on the country's earnings, employment and demographics [1]. Its mission is to become the main source of public data about the nation's people and economy. The Census data informs for instance on individual's age, level of education, employment type and income.

This information could be hugely significant to businesses. For instance, think of the businesses which target individuals with high income. These businesses could use machine learning models trained on US Census data to predict someone's income. If this person's income is higher than a given threshold (for instance $50,000 per year), then the company decides to reach out to him. Better detection and segmentation of potential customers reduces marketing costs, increases conversion rate and thus improves return on investment [2].

Although the US Census Data has been available for decades, the US Census data received much traction over the past few years/months. Some companies realized how impactful this machine learning project could be. So data science consultants started working on it [2]. However, it turns out that most of these analyses were not made on the raw US Census Data Set. Instead, most relevant past work had been conducted on an extremely simplified version of the Census Data Set: the Adult Data Set.

Goals

I was amazed to discover that real-world machine learning models, meant to inform business decisions, were based on a crazily simplified version of the data! Instead of using the raw US Census Data Set, companies used the much simpler Adult Data Set [3]. In the Adult Data Set, most features have been deleted (16 features instead of 41). This undermines the US Adult Data Set ability to grasp trends and insights. Also, training examples have been ridiculously decreased (30,000 examples instead of 300,000). And the binary target variable (whether or not people earn more than $50,000 per year) has been artificially balanced (75%/25% imbalance instead of 94%/6%)! As a result, the Adult Data Set is extremely simple to work with. But this simplification might have come at the cost of losing some valuable information along the way.

Working on the raw US Census Data Set - instead of the Adult Data Set - requires to handle more features, more training examples and to deal with extremely imbalanced classes. But this effort might lead to better performing machine learning models, that grasp more trends and insights in the data.

Therefore, my goal is to use several machine learning models to make predictions about individuals' annual income, based on the raw US Census Data Set (demographic and employment variables).

This novel data set will require new approaches, especially to deal with the extremely imbalanced classes. If it turns out that my model performs better than previous models (that were trained on the simplified US Adult Data Set), then my novel project will be very significant to many businesses.

II. Data Pre-Processing

Data

The raw US Census Dataset is a public dataset containing 41 employment/demographics variables for 300,000 individuals. I chose to focus on the raw 1994-1995 Census Data Set, because most of the relevant past work had been done on the simplified 1994-1995 US Adult Data Set: by working on the raw 1994-1995 Census Data Set, I will be able to compare my results to the previous simplified models.

The target

I tackled the following binary classification problem: based on the 41 employment/demographics variables of the US Census Data Set, how well can I predict if someone earns more than $50,000/year? In order to convert the raw data to useful features, I went through a series of pre-preprocessing steps.

From Qualitative Data to Quantitative Data

In the US Census dataset, 34 of the 41 attributes are qualitative. They contain a lot of information. However, most implementations of machine learning models cannot handle qualitative data. So the first step was to convert all non-numerical/categorical data to numerical data. This was done in two ways:
- label encoding: categorical variables that take n different values are converted into one quantitative variable taking values 1...n. This is computationally efficient since only one feature is created. But it introduces an order in the data that might not make sense in the real world. For instance, the country of birth of each individual (string) is replaced by a number from 1 to 182 (182 countries appear in the dataset)
- dummy (one-hot) encoding: categorical variables that take n different values are converted into n-1 indicator variables (one indicator variable per value). If n is large, this can

increase considerably the number of features. However, contrarily to label encoding, it does not introduce any kind of bias in the data. Note that I created n-1 (instead of n) dummy variables in order to avoid the dummy variable trap [4]. For instance, I converted the variable indicating marital status (married, divorced, separated, widowed, single) into 4 indicator variables.

Standard Pre-Processing

I then deleted the meaningless features - with 0 variance - since they offer no predictive power. The missing values were converted to aberrant values. And I normalized all the data to have mean 0 and standard deviation 1. Additionally, the target variable was turned into a proper binary (0/1) variable.

Note that I could have opted for alternative strategies to handle missing data [5]. For instance, replacing missing values by the mean or the most common value. Predicting missing values. Or simply use classifiers/implementations that can handle missing values.

Covariate Shift in Train/Test Split?

The training set consists of 200,000 examples. The remainder (100,000 examples) is for testing. The first key step was to make sure that there was no covariate shift in the data. Most supervised learning models make the assumption that training and test data follow the same underlying distribution. However, this assumption might turn out to be wrong and undermine many models. I designed the following experiment to detect a potential covariate shift.

I added to the training and test data a train/test variable indicating whether they came from the train or test data set. I then built a model trying to predict this train/test variable. If there were a covariate shift, then this classifier should be able to distinguish the train data from the test data.

The accuracy of this classifier was 66.6%. This corresponds to the baseline (66.6% of the data was training data). Therefore, I can confidently affirm that there was no covariate shift in the data.

III. Feature Engineering

Computing New Features

Based on the raw (pre-processed) variables of the data set, I computed basic features that will help the classifiers. For instance, I defined the net capital as the difference between capital gains and capital losses. This increased the number of features to 221. To judge the quality of feature engineering, I could look at the ranking of the top features of my classifiers: the net capital - a feature I engineered - ranked 3^{rd} out of 221. Success!

Feature Selection

The previous steps increased the number of features from 41 to 221. This enriched the raw dataset. But it considerably increased computation time, especially for complex models. And too many features may lead to overfitting. I therefore implemented feature selection to decrease the number of input features. To keep only the most performant features, I ran sequential backward-based feature elimination. Features were selected based on their mean-squared ROC AUC score using 3-fold cross-validation. Feature selection was terminated when this performance metric starts decreasing. This procedure led to the conclusion that the optimal number of features to keep was 171.

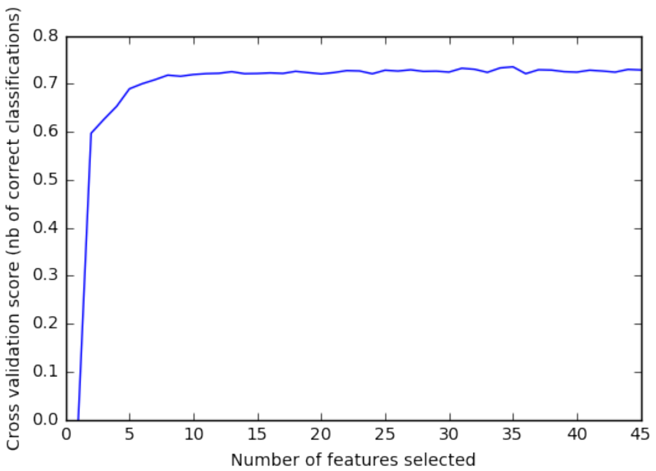

Fig. 1. Evolution of the performance of the model with the number of features

Impact of Feature Engineering

It is in general a good idea to start with a simple model that does not need much tuning - for instance a Random Forest with 30 estimators - while doing feature engineering. They are easy to implement and able to handle large amounts of variables, so they give valuable feedback on the quality of our work. Looking at the performance of the same random forest at each step of feature engineering enables us to quantify the improvements brought by each of these steps. Feature engineering increased our Random Forest's precision/recall from 0.70/0.35 to 0.72/0.39! Here are the top 5 features yielded after feature engineering and their relative importance.

Rank	Feature	Importance of the feature (%)
1	Age	0.10
2	Dividends from stocks	0.07
3	Net capital	0.06
4	Occupation code	0.06
5	Industry code	0.05

Note that among the top 10 features, 5 were not present in (or could not be engineering from) the Adult Data Set: dividends from stocks, detailed occupation recode, detailed industry recode, number of persons worked for, industry of work code and number of weeks worked in year. This is a great news for the pertinence and novelty of my project. It

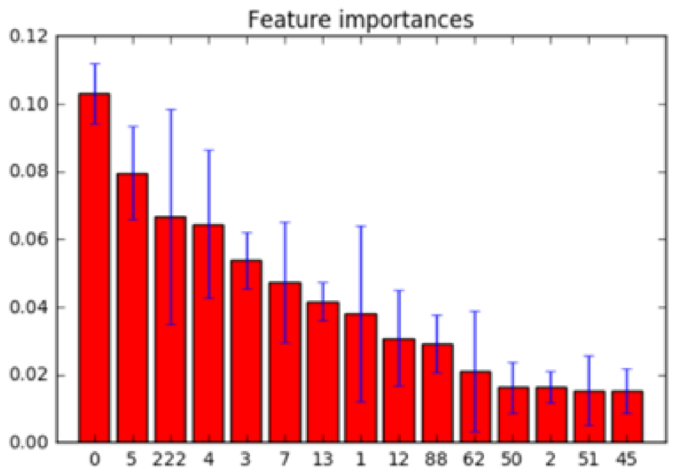

Fig. 2. Relative importance of the top 10 features

appears that by looking at the full 41 raw variables of the US Census Data Set instead of the simplified version of the Adult Data Set, we are able to grasp more patterns in the data.

IV. Fighting Imbalanced Classes

The greatest difficulty of the project lied in the extremely imbalanced classes. The target variable is binary. It is equal to 0 for 94% of the data. This forced me to implement the following methods to fight the imbalance. This difficulty distinguishes my project (on the raw US Census Data Set) from previous existing project (on the simplified Adult Data Set where imbalance was not as bad)

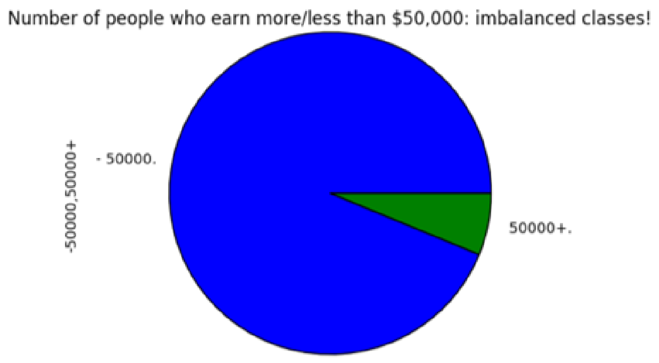

Fig. 3. Imbalanced classes

Choosing Appropriate Metrics

When classes are that much imbalanced, it is crucial to choose well the metric that tracks the performance of our classifiers. Especially, the accuracy metric is not relevant, because the accuracy baseline is 94%! Instead, I evaluated my classifiers based on their precision, recall, confusion matrices and ROC curves.

Choosing an Appropriate Cross-validation Strategy

Cross-validation can raise several issues when classes are extremely imbalanced. Indeed, there are relatively few training examples that belong to the class ”1” of the target variable (labeled as ”1”). So, if I choose the cross-validations folds randomly, the training examples labeled as ”1” might all end up in the same cross-validation fold. The other folds would be left with almost no no training example from the class ”1” of the target variable. Classifiers would struggle to learn the class ”1” on these folds. Therefore, I used **stratified** 3-Fold cross-validation to make sure that, in each fold, the proportion of 0/1 target variable is identical to that of the training set.

Introducing Class Weights

When classifiers allow it, I penalized the most frequent class, by weighting each class inversely proportional to its frequency. For instance, with the logistic regression classifier, this increased ROC AUC by 0.13!

V. Prediction Models and Methodology

In order to match the interest of companies, I formulated my learning task as the following binary classification: given the features than came from pre-processing/feature engineering, I would like to predict whether each individual has an income higher than $50,000 (the threshold $50,000 might vary from company to company).

Logistic Regression

Logistic regression is a natural choice as a first model. I ran logistic regression on the training and test set with L2 regularization. This baseline scored 0.72 ROC AUC (area under the TPR/FPP curve). I then opted for L1 regularization. When L2 regularization tended to give all the features diffuse importance, L1 regularization tended to shrink the importance of most of the features to 0, letting the classification rely mostly on a few number of features. More importantly, L1 regularization increased training time by 10x longer for comparable performance.

So I opted for L2 regularization and fine-tuned the parameter that controls regularization strength C. I ran 3-fold cross-validation with values of C in the $[0.01, 10^4]$ interval and chose the value of C that maximize the mean-squared ROC AUC cv score. This procedure found that $C = 10$ is the optimal regularization parameter. Note that, as expected, the performance of the classifier has a bell shape: when C is too low, the model is unable to grasp relevant patterns in the data (underfitting) and performance are low. And when C is too high, the performance tends to overfit and not generalize well, so performances are low too.

Note that, as mentioned in the rubric Fighting Class Imbalance, the best logistic regression model on the test set used class weights to fight class imbalance (0.13 of ROC AUC!).

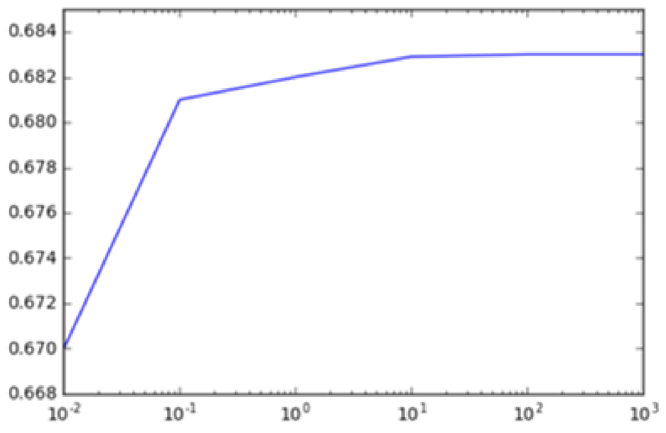

Fig. 4. Evolution of the ROC AUC (y axis) of the logistic regression with parameter C (x axis)

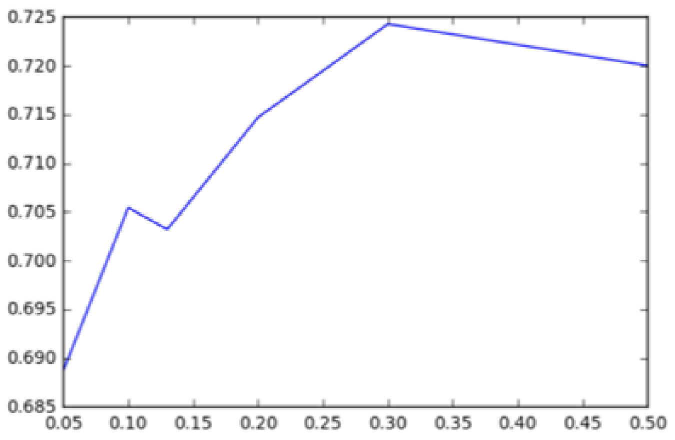

Fig. 5. Evolution of the ROC AUC (y axis) of the random forest with the parameter maxfeatures (x axis)

Having fine-tuned my logistic regression, in order to further improve the performance of my classifier, I decided to switch to tree-based classifiers. They might grasp relationships between the features and the target variable that a logistic regression could not perceive.

Random Forests

To capture patterns in the data that logistic regression might fail to detect, I opted for a model radically different from logistic regression. Random forests are meta-estimators that fit a given number of decision tree classifiers on various sub-samples of the dataset. Fine tuning random forest was made in 2 steps.

First, fine-tune the parameter that controls the number of estimators (the number of decision trees). It turned out that ROC AUC kept increasing with the number of estimators. So, my role was to find a tradeoff between computation time and performance. I decided to set the number of estimators to 100: after 100 trees, adding new trees increases the performance of the model, but extremely slowly.

Second, random forests fit decision tree classifiers based on random sub-samples of the features. I fine-tuned the parameter (maxfeatures) controlling the size of this subsample. Setting this parameter to 20% of the whole features turned out to be optimal. This does slightly better than the heuristic [6] that recommends setting maxfeature to the root of the total number of features - 13% in our case.

VI. Results and Discussion

The following figure shows the performances of my fine-tuned models. The primary error metric is the precision-recall curve: Logistic Regression and Random Forest seem to have comparable performances based on this metric. The curve might even suggest that Random Forest slightly outperform Logistic Regression.

However, looking at a second performance metric will help us understand much better the dynamics of the two models and compare them: the ROC curve. Each point of the ROC curve

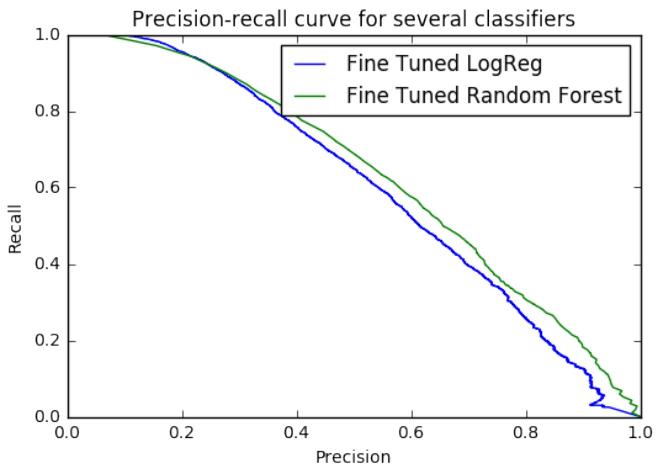

Fig. 6. Precision-recall curves of the Logistic Regression and Random Forests

corresponds to a classification decision threshold. By default, this threshold is set to 0.5. In other words, for a given test example, if $Pr(targetvariableequals1)$ is higher than 0.5, then the classifier will output 1. Setting this threshold closer to 1 improves precision, setting it closer to 0 improves recall. I decided to set the classification decision threshold to the point of the ROC curve that is closer to the top-right corner.

Finally, the area under the ROC curve leaves no doubt about the best model for our classification tast: Logistic Regression scores a ROC AUC of 0.84 when Random Forest scores ROC AUC of 0.76. Figure 7 is the ROC curve of the Logistic Regression, once fine-tuned and once it includes class weights.

VII. Ensemble Learning

In order to leverage the strengths of our fine-tuned classifiers (Logistic Regression and Random Forest), I opted for an ensemble learning method. A grid-searched Logistic Regression combined the predictions of the individual models (fine tuned Logistic Regression and Random Forest) and leveraged their strengths.

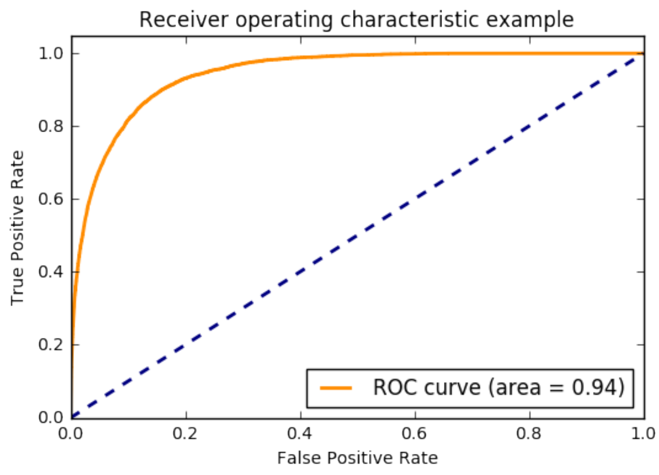

Fig. 7. ROC AUC of the fine-tuned logistic regression

The result was slightly disappointing: it improved our ROC AUC 0.841 to 0.845. Theoretically, ensemble learning methods should be able to perform much better than individual classifiers. Therefore, I see two avenues to further increase the performances of the model:
- adding new models to the stacking that will bring new insights (for instance discriminant analysis models, Neural Networks that are universal approximators)
- changing the metamodel that performs the ensemble learning. The current version uses a Logistic Regression as a meta-model. With more computation time/power, we could replace it by a more complex model (for instance the tree-based boosting classifier XGBoost)

Note: Ensemble learning methods like stacking require a validation set: I split the training set (200,000 examples) between a working set (150,000) and a validation set (50,000).

VIII. Conclusion

This elegant solution - based on rich pre-processing, feature engineering, optimization of individual classifiers and ensemble learning enabled us to increase the ROC AUC from 0.62 to 0.845. This challenging project led me to implement novel methods to deal extremely with imbalanced classes. Previous model that were trained on simplified versions of the data Census Data Set (the Adult Data Set) reached ROC AUC of 0.81. The novel improvement could be highly significant for the businesses mentioned in the introduction. It came at the expense of handling more features, more examples and extreme class imbalance, which made the project challenging.

References

[1] U.S. Census Bureau https://www.commerce.gov/doc/us-census-bureau.
[2] Predicting earning potential on Adult Dataset http://www.dataminingmasters.com/uploads/studentProjects/Earning_potential_report.pdf
[3] Adult Data Set http://archive.ics.uci.edu/ml/datasets/Adult
[4] Dummy variable trap in regression models http://www.algosome.com/articles/dummy-variable-trap-regression.html
[5] How to tune RF parameters in practice https://www.kaggle.com/forums/f/15/kaggle-forum/t/4092/how-to-tune-rf-parameters-in-practice
[6] Classification and association rules for census income data https://mathematicaforprediction.wordpress.com/2014/03/30/classification-and-association-rules-for-census-income-data/
[7] Learning Fair Classifiers https://arxiv.org/pdf/1507.05259.pdf
[8] Ballpark Learning: Estimating Labels from Rough Group Comparisons http://www.hyadatalab.com/papers/shahaf-hope-pkdd16.pdf