Machine Learning Fall 2018 Final Project Report

amc1354 & ads798

ABSTRACT

We took data from the US Census and used it to predict the yearly wage (in dollars) of a person in the country, based on other features of the person. We explored the data to understand the distribution and data type of each attribute and decided to use Linear Regression, Random Forest, and Neural Network for the task. We used as a benchmark randomly generated wages drawing from a Lognormal distribution fitted to the wages in the training data. We cross-validated different parameter sets for each model, and reduced the dimensions of the input using Principal Components in a separate experiment. We selected the best performing model wrt the Root Means Squared Error function on the cross-validation set to predicted the wages in the test data.

1 PROBLEM STATEMENT AND METRIC

The task at hand is to predict the yearly income from wages (in dollars) of a person in the United States, based on other features of the person. The data is from the United States Census (Survey, 2017). We received a train set with labels and a test set without labels. We experimented with this training set in order to develop a good model (hypothesis) using different machine learning techniques.

This metric to minimize was the Root Mean Squared Error (RMSE) function: $\sqrt{\frac{\sum_{t=1}^{N}(r^t-y^t)^2}{N}}$.

2 DATA ANALYSIS

We started with a thorough analysis of the data in the training set. Many attributes are categorical and some of them contain a large number of categories, namely `workarrivaltime`, `ancestryofparent1`, `degreefield` and `industryworkedin`. Many attributes contain missing values denoted by a '?' symbol. For every attribute, '?' has a different meaning, so we analyzed each attribute separately to decide on how to treat missing values hence if and how to impute a value. To restrict the analysis to a four-page report, we show only three examples that clarify the type of analysis we have done for each attribute and summarize the decisions we took in section 3.1.

- Attribute `workerclass` contains 10 classes, and it could be made ordinal on the basis that we expect some working classes earning more than others. We calculated the median wage by class and made a Box plot (which we omitted to keep the report in four pages). Despite the medians could group and define some ordered classes, the distribution of wages seem to overlap extensively, making difficult to see a clear differentiation by class. So we decided to one-hot encode it.

- Attribute `vehicleoccupancy` contains 11 classes, and in Table 1 we can see how many observations we have in the training data set for each category. There are only a few examples in the categories other than '1', and the training set does not contain examples for all the possible categories. '?' here means "not a worker or worker whose means of transportation to work was not car, truck, or van". We decided to group the attribute in two categories: "alone by car" (category 1) and "other".

- Attribute `workarrivaltime` is categorical with many categories. We could transform it in numerical and perhaps create one more features with group categories - e.g., "arrives from 8 AM to 11 AM" or one attributes that simply differentiates "night worker" vs. "morning worker". We could

vehicleoccupancy	no. of examples
?	775
1	373
2	30
3	4
4	1
8	1

Table 1. No. of examples by Vehicle Occupancy.

convert to numeric and set to the median arrival time where we have "?" once we stripped out the example that are non-workers. The rationale here is that "?" means either non-worker (which we removed from the data as described in section 3.1) or someone working from home so we have to impute a value for them.

For the features with many categories, we checked how many examples each categories have and we noted that most of the examples belong to just few categories. Transforming in one-hot encoding these attributes result in a very sparse dataset, so we selected models that can cope with this data (see section 3.2).
Missing values look related to jobless or minors, and the variable `workerclass` captures this definition. So we decided to predict null wage for `workerclass='?'` and train models on the remaining categories.

3 METHODOLOGY

3.1 Pre-processing and Features Engineering

Below we summarize how we decided to pre-process each variable. In general, we decided to keep one-hot encoding for all experiments, even for the Random Forest model. Decision Tree algorithms can handle categorical values, however we kept the pre-processing this way because Random Forest can cope well with high dimensionality and sparsity, and because it was simpler to compute features that contain all possible categories, i.e. that appear only on test or training data and not in the other.

- Attribute `hoursworkperweek` is used to define who's not a worker, hence have 0 wage. For this variable, '?' is defined as "less than 16 years old/did not work during the past 12 months". We checked that, in fact, all these categories have `wages` =0 and removed them from the training data. Our final algorithm predicts 0 wage for these categories in the test data. All the following decisions are made on the subset of the training data where `hoursworkperweek` is not '?'.

- We set to 0 the values of `traveltimetowork` that are '?' because they refer to who works from home, hence travels 0 time units.

- We defined a new feature, `workshift` with 3 classes: "dayarrival" for `workarrivaltime` between 8 AM and 7 PM (to take into account work habits rather than legal definitions such as 9 AM to 5 PM); "nightarrival" for `workarrivaltime` between 7 PM and 8 AM; "wfh" to indicate "working from home" for `workarrivaltime` ='?'.

- We transformed `workarrivaltime` into an ordinal, numerical variable and imputed values where there was a '?' by taking the median value of the non-'?' examples.

- For the remaining variables, and the ones described above, we computed one-hot encoders for the categorical attributes and left the numerical ones as they were. At this point all the '?'s were dealt with.

- We finally standardized all the attributes, i.e. subtracted the mean and divided by their standard deviation.

3.2 Algorithms Choice

Given the nature of the problem, we looked into the class of regression models. For this data set, there are good grounds to use any of these: Naive-Bayes, k-NN, Clustering, SVM, Linear Regression, Neural Networks, Decision Trees. We decided to focus on Linear Regression, Random Forest and Neural Network.

Linear Regression because this algorithms minimize the sum of squared errors, which is close to the evaluation metric (Section 1). One challenge we had here was dimensionality. Once transformed all the variables, we ended up with a data set of 410 attributes. Moreover, running 5-fold cross validation, we trained on a data set with 460 examples, which is a similar dimension as the number of attributes. We solved this problem by using L^2 regularization, aka "Ridge", which minimizes $||y - X \cdot w||_2^2 + \alpha \cdot ||w||_2^2$ (the higher the α, the more coefficients will be set to 0 hence the regression uses less attributes), and by adding one more experiment using dimensionality reduction thanks to Principal Components Analysis (PCA).

Random Forest - decision trees are good candidates with data sets with many categorical variables. As mentioned already, we left the data pre-processed with one-hot encoding and RF performs dimensionality reduction thanks to the way the algorithm works, i.e. by choosing a subset of variables to fit a single tree to add to the forest.

Neural Network work best with classification problems, and with problems like text processing or image recognition. However, given the high dimensionality of the data set and the highly non-linear decision function we expected, a neural network was an interesting candidate to explore.

We used the Python implementation of Scikit Learn (Pedregosa et al., 2011) for all these models. We took 80% of the training data set to experiment with different models and tune their parameters. For each model, we run 5-fold cross validation to check what was the most reasonable choice of some of the models' parameters. Then we tested the best model on the holdout 20% testing set, and reported the RMSE to show which one performs best on unseen data. We then picked the model with the smallest RMSE and fitted it on 100% of the original training set. This latter model was then used to predict the unlabeled test set and produce the output required by the assignment.

3.3 Benchmark

We decided to use a simple benchmark that represents a random choice of wages without looking at the attributes. We fitted a Normal distribution to the log transformation of `wages`, and used the parameters to draw random values. By taking the exponential of these random numbers, we have a random prediction of `wages`.

4 RESULTS

In this section, we summarize the results with the models described in section 3.2. We report the RMSE in thousands and with two significant digits to simplify reading the results. E.g., the RMSE of the benchmark was 180600.25277479065. We report this number as 180.60. In Table 2, we summarize the results with the choices of parameters and cross validation results. As expected, Linear Regression performs really bad, and it does much better when we use regularization improving the benchmark quite significantly. Random Forest is the model performing better with the lowest RMSE on the test set. Neural Network gives a worse prediction than the linear regularized model. This might be due to the fact that Gradient Descent (GD) did not converge in our experiments. We tried running the model with different values of the max number of iterations of GD, and cross-validated different learning rates, but none converged. We stopped experimenting at max 10,000 iteration of GD because it was causing the analysis to run too slowly on our laptops.

4.1 Using PCAs

To help the performance of the Linear model, we decided to pre-process variables with PCAs. We then selected the first 200 PCAs as they explained just above 80% of the variance in the data. Below we report the results on the same class of algorithms.

Model	Parameters values tried	Best parameters set	RMSE
Benchmark	NA	NA	180.60
Linear Model (LM)	NA	NA	$2.90 \cdot 10^{16}$
LM w/ Ridge Regularization	$\alpha = [100, 1000, 10000]$	$\alpha = 1,000$ (mean CV RMSE=78.94)	71.56
Random Forest	$nestimators = [100, 200, 300, 410]$ $maxdepth = [50, 100, 200]$	$nestimators = 200$ $maxdepth = 200$ (mean CV RMSE=79.24)	57.74
Neural Network	$learningrate = [1, 0.01, 0.001]$ $hiddenlayersizes = [50, 100]$	$learningrate = 0.001$ $hiddenlayersizes = 50$ (mean CV RMSE=96.69)	90.56

Table 2. Experimental Results. Note the RMSE is calculated on the test set hold out taking 20% of the original training data. The metric used to calculate the best parameters set is the average of the RMSEs of the 5 holdout (validation) sets taken from 80% of the original training data during the 5-fold cross validation runs.

Model	Parameters values tried	Best parameters set	RMSE
Benchmark	NA	NA	180.60
Linear Model (LM)	NA	NA	154.72
LM w/ Ridge Regularization	$\alpha = [100, 1000, 10000]$	$\alpha = 1,000$ (mean CV RMSE=79.38)	72.20
Random Forest	$nestimators = [10, 50, 100, 150]$ $maxdepth = [10, 50, 100]$	$nestimators = 100$ $maxdepth = 150$ (mean CV RMSE=88.08)	76.91
Neural Network	$learningrate = [1, 0.01, 0.001]$ $hiddenlayersizes = [50, 100]$	$learningrate = 0.001$ $hiddenlayersizes = 50$ (mean CV RMSE=103.48)	92.27

Table 3. Experimental Results using PCA.

As we expected, we saw a big improvement in the Linear Model without regularization though it was a little improvement over the benchmark. It is interesting to note that for all the other models that are able to deal with high dimensional datasets, the RMSE worsen a bit for regularized Linear Model and Neural Network, and significantly for Random Forest. This might be due to the loss of information that we operated by choosing the first 200 PCAs.

5 CONCLUSION AND FURTHER WORK

We analyzed data from the US Census to predict wages. We run a thorough analysis on the training data that informed us on how to pre-process the attributes. We included one experiment transforming the variables with Principal Components. We chose to explore a Linear Model, a Linear Model with Regularization, a Random Forest and a Neural Network. We tuned the models' parameters running 5-fold CV on 80% of the training data. For each model, we chose the best performing set of parameters (according to the mean RMSE over the 5 holdout samples), fitted it on 80% of the training data and evaluated the RMSE on the remaining 20%. The best performing model was Random Forest with 200 estimators and 200 max depth. This model was then fitted to 100% of the training data to build predictions for the unlabeled test data.

It would be interesting to try different pre-processing and features engineering. E.g., manually group data in categories that make common sense, convert into ordinal numbers categories that display ordinal correlation with wages, add features using clustering methods such as k-Means or k-NN to group similar workers into new categories. More experiments shall be run on the Neural Network by increasing the number of iterations of GD and introducing more layers. Other algorithms such as SVM and k-NN would be interesting candidates (hypothesis) to explore for this data.

REFERENCES

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Survey, T. A. C. (2017). The data was adapted from The American Community Survey. it is only a very small part of the data, and it was preprocessed by the course's instructors for use in this assignment. *https://www.census.gov/programs-surveys/acs/technical-documentation/pums/documentation.html*.