MEBDI ML Competition Report

Lejvi Dalani Hasan K. Tosun

Part 1

Data

We downloaded the train and test datasets from MEBDI's website. All input variables except age are categorical.

Algorithm

For this part of the competition, we tried a number of Machine Learning techniques to minimize the test error. However, OLS with transformations of the age variable and factor interactions slightly outperformed all the ML algorithms we tried. We used polynomial and log of age, and interactions of them with other variables. The mathematical expression for the regression is:

$$\begin{aligned} log(wage) = \alpha &+ \beta_1 age + \beta_2 age^2 + \gamma_2 log(age) + \gamma_2 (log(age))^2 + educ\_cat + race\_cat + statefip + male \\ &+ educ\_cat * age + educ\_cat * (age^2) + male * age + male * (age^2) + educ\_cat * male \\ &+ educ\_cat * male * age + educ\_cat * male * (age^2) + educ\_cat * race\_cat + race\_cat * \log(age) \\ &+ race\_cat * ((log(age))^2) + male * \log(age) + male * ((log(age))^2) + race\_cat * male \\ &+ race\_cat * male * \log(age) + race\_cat * male * ((log(age))^2) + statefip * male + \epsilon \end{aligned}$$

Results

In this section we report the test sample results for our algorithm for part 1. The algorithm was trained using only the training sample with log wages as the target variable. The results show the error on the test sample using the three metrics required for the competition.
Our results for this part of the competition for the **test sample** are:

RMS-log	**RMS-level**	**AbsDev-log**
0.715935794374381	30578.7632105822	3.66628123170716

Note:

1. **RMS-log** is calculated as:

$$\textbf{RMS-log} = \sqrt{\sum_{i \in \mathcal{T}} (y_i - \hat{y}_i)^2}$$

where $\mathcal{T}$ denotes the test sample, y_i denotes the test target (natural log of wage earnings), and $\hat{y}_i$ the predicted value.

2. **RMS-level** is calculated by replacing the logs with levels of income in the equation above.

3. **AbsDev-log** is calculated as:

$$\textbf{AbsDev-log} = \max_{i \in \mathcal{T}} |y_i - \hat{y}_i|$$

where $\mathcal{T}$ denotes the test sample, y_i denotes the test target (natural log of wage earnings), and $\hat{y}_i$ the predicted value.

Part 2

Data

- We downloaded all the variables from CPS 2012 from IPUMS.
- We divided the variables into 10 categories:
  - Technical variables: These are the variables that are created by IPUMS or Census Bureau. Some examples are IDs to keep track of the individuals, families and households, and weights associated to them.
  - Unused disallowed variables: These variables are not allowed to be used as they are mechanically related to the outcome variable. Some examples are hours worked, and weekly average wage.
  - Variables with a single value: These variables has only one unique value in the whole dataset. An example can be *cpi*99.
  - Indirectly-used disallowed variables: There are some variables that are disallowed due to being created by using the income. We use those variables to create new variables with very coarse categories. For example, we transform the amount of housing subsidy to a 0/1 variable; 0 if the subsidy amount is zero, 1 if the subsidy amount is positive. We call the original housing subsidy variable "indirectly-used disallowed variable".

- Indirectly-used allowed variables: There are some variables that are allowed, and that we use them to create new variables with some manipulation. One type of manipulation is used on the variables with too many categories. To reduce the number of categories, we group some of the categories in those variables together. For example, there is a variable for the number of families in the household. We created a new variable; it is equal to its original number if the value is less than 3, and it is equal to 3 if it is greater than or equal to 3. Another type of manipulation is to mutate new variables using the information from an existing variable. For example, we generate a variable $haskidsunder3$ by using the information in *yngch*, age of the youngest child. So we call the original variable *yngch* as "indirectly-used allowed variable".
- Allowed categorical variables: There are some allowed categorical variables in the dataset that are allowed to be used. *sex*, *education*, $statefip$ are among them.
- Allowed continuous variables: There are some allowed categorical variables in the dataset that are allowed to be used. *spmmedxpns* and *spmchxpns* are among them.
- Imputed variables using disallowed variables: These are the variables that are created using the indirectly-used disallowed variables in a way that is described above. $houssub\_$ is an example.
- Imputed variables using allowed variables: These are the variables that are created using the indirectly-used allowed variables in a way that is described above. $ncouples\_$, $vietnam\_$ and $whymove\_$ are among them.
- Imputed variables using family interrelationships: We use the data from other family members to create new variables. The only variable in this category, $spwage\_$, is the (log of) the wage income of the spouse of the individual. There is not a mechanical relationship between the person and their spouses' wage, so we decided to create that variable.

In our final dataset, we only used the variables from the following categories: Allowed categorical variables, allowed continuous variables, imputed variables using disallowed variables, imputed variables using allowed variables, and imputed variables using family interrelationships.

Variables Used

We provide a list of variables in `var_list.csv` with the variable names, decisions and whether they are used in any part of the project. If decision is `drop`, it means that the variable is not used as it is in the algorithm. The other categories for `decision` are created for the algorithm. There are two possible values for `used`: `yes` means it is used in some part of the project, and `no` means

otherwise. There are some variables that are dropped but used. It means that those variables are not used as they are; the transformed versions of them are used instead. We described the transformations in the project report.

Algorithm

KNN Imputation of the Missing Values

As we described in the previous section, spouse's wage (*spwage*) is created using the variables containing the family interrelationships and the *incwage* of the spouse. However, the variable is not complete; there are some observations with missing values. To complete the values in that variable, we used K-Nearest-Neighbors algorithm to impute the missing values of *spwage* by using the other features that go into our final algorithm. It is important to note that we didn't use the output variable in the imputation process.

Entity Embedding of Categorical Features

Variables *occ* (occupation) and *occly* (occupation last year) showed up as important features across multiple feature selection algorithms we tried. However, both features had high dimensionality (more than 500 categories in each), with very few observations for some categories in some of the samples. Given how small our sample size was, we decided to reduce the dimensionality of the two variables. We did this by training a Neural Network: the first two layers were embedding layers of size 23 each (approximately the square root of the categorical size of each variable), the next two layers were hidden layers of size 2000 and 10 with ReLU activation, while the final layer was a linear activation layer. The Neural Network described above tried to train and predict the *log wage* variable on the *occ* and *occly* variables, and in doing so we were able to infer relationships based on occupation and occupation last year and wage. This allowed us to reduce the dimensionality of each of the occupation variables to 23. The idea of entity embeddings is to create vectors in a smaller subspace (in this case in R^{23}), instead of the sparse R^{500+} space needed by dummy encoding. Each vector in R^{23} corresponds to one unique occupation. As a consequence, *occ* and *occly* were mapped into 46 columns. These 46 columns were merged into our dataframe, and the raw *occ* and *occly* variables were dropped, in order to avoid overfitting, since our train data had already seen the two variables.

We decided to perform two types of encoding on the remaining categorical variables in our data: $LabelEncoding$ (randomly assigning numbers to each category within a variable) and $OneHotEncoding$ (creating dummies for each variable). These encodings were performed uniformly on all categorical variables in order to create three dataframes: DF_{enc}, DF^1_{onehot} and DF^2_{onehot}, where DF^1_{onehot} and

DF^2_{onehot} are randomly column partitioned (of equal size) from DF_{onehot}, a dataframe resulting from OneHot encoding all of the categorical variables. The partitions will become useful in our discussion that follows regarding the algorithm.

Stacked Generalization

We used Stacked Generalization as the algorithm for the competition. Our Stacked Generalization algorithm consisted of two levels:

1. 1st Level
2. Meta Level (2nd level)

1st Level layer:

The 1st Level consisted of:

- Algorithms (tree-based) trained on the label encoded data, DF_{enc}:
  - A Random Forest algorithm, with off-the-shelf parameters
  - An XGB Regressor, which we tried to optimize using Random Search Cross Validation over the parameter space
  - A Gradient Boosting algorithm, with off-the-shelf parameterization, with a few tweaks to learning rate and depth of the trees.
  - An LGBM Regressor with off-the-shelf parameterization with smaller learning rate
- Algorithms (linear) trained on the OneHot encoded data, DF^1_{onehot} and DF^2_{onehot}:
  - On each of the two column partitions of our data, the same Lasso regression was applied, for a total of two models. The regularization parameter was fine tuned based on cross validation using the training data.

The reason for the two types of algorithms mentioned above is the following: tree based algorithms perform better on non-dummy variables, while linear models perform better with dummy-encoded variables. Since there are disadvantages to both types of encoding with respect to training models, we decided to control for both, and thus increase the diversity of our model. The predictions from all the models in the 1st level were then stacked together.

Meta Level layer:

The stacked predictions from the first layer were appended to our data again. In this layer, we turned

all categorical variables into strings, and tried to spoil CATBOOST algorithm's ability to handle categorical variables. The dataset we fed into CATBOOST consisted of the original data (with string categorical variables) and the six columns coming from the 1st level stacked predictions. We used an off-the-shelf parameterization, and fine tuned the number of iterations, the learning rate, and the depth of the algorithm to what we thought would be a reasonable level, since Random Search results were discouraging. We appended our data to this layer at the hopes that CATBOOST would capture some features of the categorical nature of our data that label encoding and OneHot encoding may have missed.

The two layers combined make up our prediction model.

The following box summarizes the main steps for the algorithm we used for this part of the competition.

1. **Trained a Neural Network with Entity Embedding Layers for *occ* (occupation) and *occly* (occupation last year) to reduce the dimensionality.**
2. **Used Stacked Generalization to train our data in two levels**
   - **In the 1st Level (StackNet with four folds)**
     - Trained four tree-based models on the label encoded variant of our data
     - Trained a linear model on the two column-partitions of our one hot encoded data
   - **In the Meta Level**
     - Trained a CATBOOST algorithm on a dataset consisting of prediction from the 1st level and our data
3. **The Stacked Generalization model was used to make predictions on the test data**

Results

In this section we report the test sample results for our algorithm for part 2. The algorithm was trained using only the training sample with log wages as the target variable. The results show the error on the test sample using the three metrics required for the competition.
Our results for this part of the competition for the **test sample** are:

RMS-log	RMS-level	AbsDev-log
0.4783689841990681	22531.036588500036	3.2039062708946187

Note:

1. **RMS-log** is calculated as:

$$\textbf{RMS-log} = \sqrt{\sum_{i \in \mathcal{T}} (y_i - \hat{y}_i)^2}$$

   where $\mathcal{T}$ denotes the test sample, y_i denotes the test target (natural log of wage earnings), and $\hat{y}_i$ the predicted value.

2. **RMS-level** is calculated by replacing the logs with levels of income in the equation above.

3. **AbsDev-log** is calculated as:

$$\textbf{AbsDev-log} = \max_{i \in \mathcal{T}} |y_i - \hat{y}_i|$$

   where $\mathcal{T}$ denotes the test sample, y_i denotes the test target (natural log of wage earnings), and $\hat{y}_i$ the predicted value.