ORIE4741 Learning with Big Messy Data
Fall 2020

Midterm Report

Mental Health Study in Tech Industry

by
dh734 - Donghao Huong
lc977-Lihe Cao
ss2742 - Sukriti Sudhakar

1. Data Description

The dataset we are using is in an SQL file. The data is separated into three sheets. They are Answer sheets, Question sheets, and Survey sheets. The Survey shows that this organization used five questionnaires from 2014 to 2019. The Questions collected all the questions asked in these five questionnaires and the Answer sheets record all the answers of these five questionnaires. We find that there are 4218 observations and 108 features in this dataset. Comparing the questions of these five questionnaires, we find that the questions differ in years but are the same from year 2017 to 2019. In the project, we only focus on the observation from these three questionnaires. Therefore, we finally used data with 1524 observations and 78 features. Among these 78 features, 57 of them are numerical data, 6 of them are text data, and the rest of the data are boolean data and nominal data. In this report, we do not include the feature text first. We plan to use the NLP tool to process these features in our following part of the project. In the next part, we will use encoding method to assign values to the nominal and ordinal data. Then, we will fill up the missing values in them.

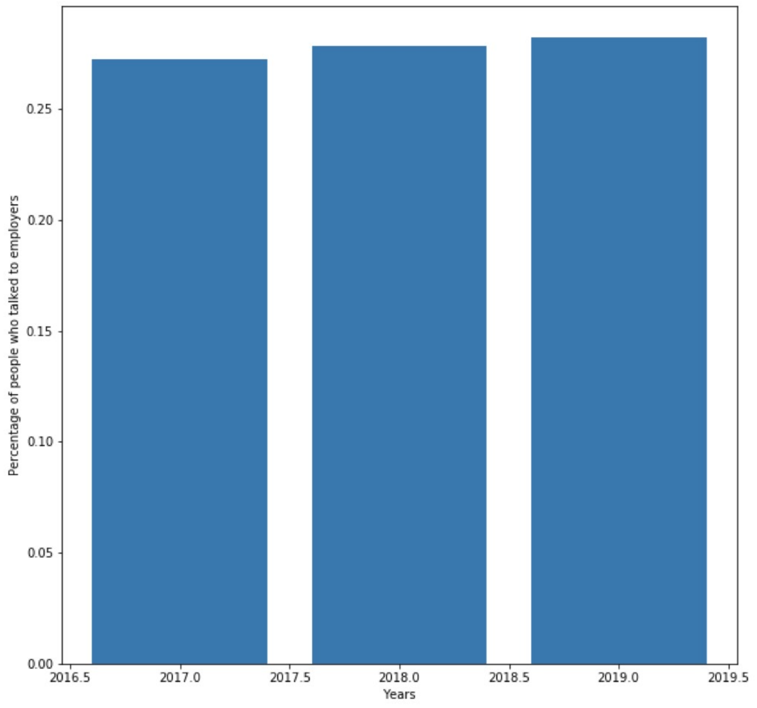
Data Visualization

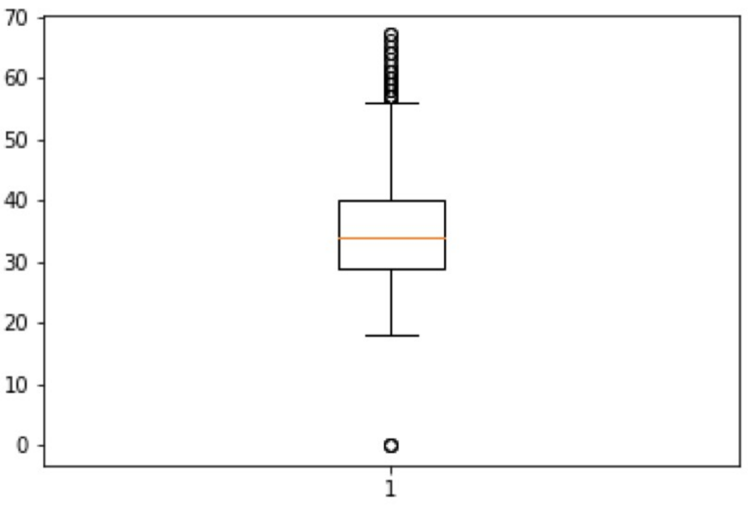
(1) Willingness to discuss mental health issues with employers and coworkers

We see in the bar plot given below that the percentage of people who talk about their mental health issues has continuously increased over the time of the

(2) age of box plot

The y axis of the box plots is the age of the volunteener. We can see in the graph of box plots, most of the people took questionnaire is 20 to 55 year old.





2. Data Preprocessing

(1) Nominal and Ordinal Data Encoding

For the boolean and the nominal data, we use the ordinal encoding for each response. For each column, we first count each label first, then sort them by the frequency descendingly. Then we assign the value to each label by using the order we got from the previous step. As a result, the data with a higher apparent frequency will have a higher value. There are two reasons we choose to use the ordinal encoding. The first one is that the ordinal encoding is the most straightforward. There are 32 features that need encoding. We can quickly assign value to each answer in the short time. The second one is that, unlike the one-hot encoding, the ordinal encoding will not increase the size of the whole dataset. Therefore, the data will not become super large and requires a large amount of computation power. Besides the computation power, the large amount of features will also probably make the model become overfitting. However, the data we are using are not ordinary data. Our encoding method may be too straight forward and makes our final model become too simple and consequently underfitting. We may use other ways of encoding these features in the following part of the project.

(2) Fill up missing values

In our raw data, there are many mssing values. There are 19 questions which have more than 1100 missing values, where other questions have 223 missing answers at most. For now, we just throw them away these questions which have more than 1000 missing values and use the remaining features to fit our model. We will consider how to deal with these features later in the project. We then need to make up the missing values in the remaining features. There are three kinds to values, one is numerical values from in a certain range like poeple's age, one is ordinal values like the answers to the question "how much importance does your imployer place on physical health?", and the last one is bolean values "0/1" or "-1/1" transfered from "Yes" or "No" answers.

There are several ways to fill up the missing values for these kinds of data.

(a) one of the simplest and widely used way is to take the mean of the remaining data assign it to the missing value. If the data of a feature is integer valued, we should round it first.

(b) for the bolean values "0/1", we can calculate the mean of the data, which is a number in the interval [0,1], we then use the mean to be the probability for a missing data to have value 1 and use simulation to fill up the missing value. For example, if the mean is 0.6, we can generate a uniform random number in [0,1], if it falls in [0,0.4], we set the missing value to be 0, otherwise, we set the missing value to be 1.

(c) for each feature, we can take it out and let it be the output we want to predict and let other features to be the input examples. We can fit a model for this feature use the data we have and predict the missing value using this pre-trained model. In this way, we can fit a model for each feature and fill up the missing values.

In our project, we will fill up the missing values by calculating the mean of data of each feature before midterm and use this data to fit our final model to see the prediction result. If the result is reasonable, we will using other methods to fill up the missing values in the last part of our project to see whether we can refine our model.

(2) Separate Training and Test Sets

We get the training example and test example sets from the original dataset using "thumb rule". We first shuffle the data matrix and then take the first 80% of the data to be training examples and the rest 20% to be the test set.

3. Model Fitting

(1) Model and Parameter

Our project goal is to predict whether the interviewee has ever sought for a mental treatment disorder from a mental health professional, which has bolean values 0 or 1. Therefore, we choose to use logistic regression and SVM as our models which are two most widely used models in solving classification problems. We then use "sklearn" packcage in Python to fit the model using our pre-processed data. Before the midterm, we have mainly used logistic regression to do the prediction. We have also tried to use the data to fit the SVM model. In the logistic regression model, we are going to solve the optimization problem

$$max\Sigma_{(x,y)\in S}log(1+exp(-yw^Tx))$$

We tried to add different regularization terms in the model. We fit the model by adding l_2 regularization term, l_1 regularization term and no regularization term. We also tried different solver such as 'liblinear', 'lbfgs' and 'saga' to see which slover gives better result. When running the algorithm, we set the maximum iteration to be 100000 times.
In the SVM model, we are going to solve the optimization problem below,

$$min\Sigma_{(x,y)\in S}l_hinge(x_i,y_i;w)+\lambda||w||^2$$

Before midterm, we didn't adjust the parameters. **(2) Prediction Result**
(a) Logistic regression Here we will show some of the accuracy rate for the model using different regularization terms and solvers. We train the model using the training examples and get the accuracy rate using the test set.
solver: 'lbfgs', regularization term: 'l_2', accuracy of training set: 84.49%, accuracy of test set: 80.66%
solver: 'liblinear', regularization term: 'l_2', accuracy of training set: 88.51%, accuracy of test set: 82.29%
solver: 'liblinear', regularization term: 'l_1', accuracy of training set: 85.89%, accuracy of test set: 78.03%
solver: 'saga', regularization term: 'l_2', accuracy of training set: 75.88%, accuracy of test set: 74.09%
solver: 'saga', regularization term: 'l_1', accuracy of training set: 79.08%, accuracy of test set: 75.74%
(b) SVM
We only fit the model using 'Radial basis function' as kernel function. The accuracy of training set: 99.02%, accuracy of test set: 79.67%. This is expectable since we didn't add penalty to the loss function, the error of the model applying on training set should be close to 0. Later in the project, we will try different kernels in the SVM model and adjust parameters such as the coefficient of the penalty and the kernel function and compare the results with the logistic regression model.

(3) Overfitting and Underfitting

In the previous part, we used our model to fit both the training set and the test set and calculate the accurate rate. We can observe that the accurate rates for predicting training examples are around 80%, which are quite high, so the model is not underfitting. We then compare the accurate rate of the test set with the training set, using different solvers and regularization terms respectively. From the results above, we can see they are very close. For example, when we use 'lbfgs' solver and 'l_2' regularization term, the accuracy of training set is 84.49% and accuracy of test set is 80.66%. The difference is: $\frac{\text{84.49\% - 80.66\%}}{\text{84.49\%}}=4.53\%$. The difference is small enough for us to claim that the model is not overfitting.