Attached file: HW5.ipynb

I open the question data from the HW5.csv file as a pandas dataframe. These data are a total of 32561 rows of data, each of which is made of 15 features. 6 of which are numeric values and the rest are strings.

There are 1836, 1843 and 583 missing values on 3 of these columns (WorkClass, Occupation, NativeCountry) respectively. There are several ways to deal with missing values. All 3 columns have group qualitative values. Among the most inefficient (!) methods of dealing with missing values is deleting rows or (worse!!) columns. A better method, which works for qualitative data but is not very smart, is to use the mode value of each feature to fill in the missing value. It means to use the most frequent label of each column instead of the empty value. The strength of this method is that the data mode remains constant. Since I am dealing with statistical categories in the following, in a better case, efforts should have been made to maintain the central parameters such as median and average, but since the data are not numerical now, it is not possible to use the median and average. On the other hand, more intelligent methods like KNN require a lot of pre-processing. Another problem is that to solve the problem of missing values, the test and training data should be separated in the best case. In using both methods (KNN and Mod), the test data is affected by training, but because KNN itself is a classification method, the impact of the final results will be greater. I plan to report the results instead of separating the test and training. I use cross_val_score with the accuracy criterion from the sklearn library, so that the pre-processing performed on the entire data has less impact on the final result, I use the same method of inserting the most repetition or the same mode of each column.

On these 15 features of the last column, or Income, we consider the dependent variable or the target, which either has a value of <=50K or >50K. To continue the work, I will take these tags in the order of zero and one.

I actually have an attribute on the Education and Education-num columns, which is in the first column as a string value and in the second as a coded number. To continue working, I continue with the coded column and because the value of this qualitative column is ordinal, that is, the increase and decrease of the code value is meaningful (low and high level of education), I can directly use this column to build the model without further pre-processing. do

So, my total independent columns are 13, 6 of which have numerical values, one has an ordinal value, and the other 6 are categorical, which require other pre-processing and must be coded. The coding of these columns should include the following steps:

1. Identify the total number of states on each column. Suppose d state.
2. Consider each state as a variable. That is, we take d up to the new column.
3. For the value of that column, the variable equal to the value of that column should be set to one and the rest to zero.

4. To eliminate the dependence of the added columns, the first column should be deleted because by having d-1 columns, it is possible to find whether column d is 1 or 0.
5. On the main data, instead of this qualitative column, I will replace the generated binary column d-1.

This operation is performed on all 6 columns at once using OneHotEncoder from sklearn.preprocessing and the resulting data has 82 columns as an independent vector. The pre-processing that deals with the scale of the data does not have much effect on the regression model because the coefficients of the regression model can be selected according to the scale of the data. For the simple Bayesian model, I also want to use the Gaussian distribution for all columns, which estimates the distribution of the mean value and variance from the data itself with the ML method and uses it as a parameter, and finally, the values are based on the probability distribution and the classification result. It is not effective, so without doing additional pre-processing, I use the same data for all continuation models and take the values of df['y'] as the two-mode dependent column.

As a base state classifier, I use a clfB single-state classifier that labels all data with a value of zero or something less than 50k. The test result is equal to the relative frequency of zero class on the test set. which is obtained from the cross validation method with cv=10 times, the average accuracy is about 76%.

As the second category, I use clf1, which is a linear regression model. I use LogisticRegression, the same sklearn library, with the default parameters of penalty = 12 and maximum repetition equal to 100. With the cross validation method with cv=10 times, it gives an average accuracy of nearly 80%.

The last model I use is clf2, which is a simple Bayesian classifier that assumes a Gaussian distribution for all columns. I use simple Bayesian models from GaussianNB from the sklearn library. With the same criteria and method as before, this model gives an accuracy of nearly 80%.

The accuracy numbers obtained for each model are detailed below. Both clf2 and clf1 models clearly work smarter than the single-mode model, which means that learning on these data is fruitful and they perform close to each other, but the logistic regression model has a slightly better result.

classifier	**method**	**accuracy**
clfB	F(x) = 0, constant	**0.7591904489970196**
clf1	Logistic regression	**0.7969043240074865**
clf2	Gaussian Naive Bayes	**0.7951538040538655**