CIS 662: Assignment 3

Instruction: This is an individual assignment. You are welcome to look anything up online but do NOT collaborate with your classmates or get significant help from anyone outside. The first two parts are IN-CLASS and the last three are TAKE HOME. The due date for the take-home is a week from when this is assigned. For both in-class and take-home, you need to respond to the questions on the online portal and upload your code with outputs as a single PDF document.

1. **In-Class – Get data and pre-process**: Download the file invistico_Airline.csv which is available on Blackboard and is nearly 13MB. This file is based on an airline survey that passengers filled out. The columns of the data set are fairly self-explanatory, some are opinions filled out by the traveller, some are flight-related data, and others are demographics. Two columns that will be significant shortly are `'satisfaction'` and `'Class'`. By `'satisfaction'` it means whether or not customers were satisfied with their flight experience. By `'Class'` it means whether the passenger flew business class, economy class, or economy plus (this is between business and economy).

   (a) Read the CSV file into your python code, print the header (i.e. the usual first 5 lines), and also print the names of the columns.

   (b) Next, check if there are any `NaN` values in any of the files, and drop those lines (the python command for that is `dropna()`, you can look it up). *Which column had NaN values and how many were there?*

   (c) *State the number of rows (n) at this stage in the single dataframe without counting the column name.*

2. **In-Class – Binary Logistic Regression**: This is similar to what we did in class, so feel free to use that code. For this section use the dependent variable (or Y variable) as `'satisfaction'` which is a binary variable.

   (a) How many features (independent variables) are in the data set? Which of them are categorical?

   (b) Do a one-hot encoding of the dataset, make sure to drop-first, and explain how many features are there after the encoding?

   (c) Split the data into train and test by randomly selecting 20% of the data to be in the test set (and use stratified sampling) with random state of 50. *Write down the length of the train and the test data (as well as the first row's index for both). Be sure to not use a comma for numbers greater than 1000.*

   (d) Scale the data using standard scaler like we did in class. Perform a binary logistic regression using the train data. Make sure to use `lbfgs` as the solver, set multi class as `auto`, and no penalty. *Write down the score (fraction of accurate predictions), as well as $\hat{\beta}_0$, $\hat{\beta}_1$, $\hat{\beta}_2$ (but not the others). Comment on the performance.*

   (e) Create a dataframe to predict the `'satisfaction'` variable with the given features in the test set using the independent variables. Also include the ground truth (actual `'satisfaction'`). *Write down the accuracy correct to 2 decimals.*

(f) For the test predictions dataframe, add a column for the probability that `'satisfaction'` is 1. *For the first index value of the test data frame, write down the probability correct to 2 decimals.*

3. **Take-Home – Multinomial Logistic Regression**: This is similar to what we did in class, so feel free to use the same code. For this entire take home part, the Y variable (i.e. dependent variable) is `'Class'`. Many times when one fills out surveys such specific details are not filled out as it may reveal the identity. So we wish to be able to predict the class. Drop the column `'satisfaction'` (you are welcome to leave it in too but be sure not to use it as a feature to predict).

   (a) Start with the original data and drop NaN like we did before. Be sure to *not* perform one-hot-encoding for `'Class'`. But perform one-hot-encoding for the other independent variables. How many independent variables are in the dataframe after performing one-hot-encoding?

   (b) Split the data into train and test by randomly selecting 20% of the data to be in the test set (and use stratified sampling). Also make sure to use random state of 50.

   (c) Do NOT scale the data yet. But feel free to write the code using standard scaler like we did in class, and put an `if False:` over it. Perform a multinomial logistic regression using the train data and the response variable Y as 'Class' considering the OVR model. For the multinomial logistic regression use solver as `lbfgs` and no penalty. *Write down the score (fraction of accurate predictions), $\hat{\beta}_0$, $\hat{\beta}_1$, $\hat{\beta}_2$ for the train data.*

   (d) If the algorithm does not converge in 100 iterations, one by one try one of these and reset the others to the original case above. Try to (i) increase the number of iterations to 1000, (ii) use a scaler, (iii) change the algorithm to newton-cg. Do not use a penalty here. *In each of the 3 cases, what was the train data accuracy?*

   (e) For each of the 3 cases above create a dataframe to predict the `'Class'` variable with the given features in the test set using the independent variables. Also include the ground truth (actual `'Class'`). *Write down the test set accuracy correct to 2 decimals.*

   (f) For the code output you only need to use case (ii) above which is scaled data. Also, for the remainder, please use scaled data.

   (g) For case (ii), *which class was predicted wrong the most number of times in the test set? Also which was predicted wrong the fewest time?*

4. **Take-home – k-neighbors method**: This dataset is not the most ideal for K-Neighbors classifier for its large size (so running it will take a long time) but otherwise it is ideal for some reasons including that the number of each of the multiple classes are similar (and not lop-sided), the number of features are not too many, and all the features seem to be useful. It is expected that the method would perform well, but it takes time to run.

   (a) Read up `sklearn.neighbors.KNeighborsClassifier` to learn how to use it. Use all default values, except the number of neighbors. For number of neighbors use 7. Use the same testing and training as before, and also use scaler.

(b) Do the training to fit. *Write down the score (fraction of accurate predictions) for the train data.*

(c) Create a dataframe to predict the `'Class'` variable with the given features in the test set using the independent variables (of course, you should not use 'satisfaction'). Also include the ground truth (actual `'Class'`). *Write down the accuracy correct to 2 decimals.*

(d) Make sure this method is also part of your code that you upload with outputs.

5. **Take-Home – Multinomial Logistic Regression with Penalty**: The reason we did not do this before k-neighbors is because we are going to subset the data for this part. Start with the main data set and remove the NaN values like before. Then, use only the rows where the `'Inflight entertainment'` is equal to 0. This should leave you with 2968 rows. Split the data into train and test by randomly selecting 20% of the data to be in the test set (and use stratified sampling). Also make sure to use random state of 50 and do not use satisfaction, class, or inflight entertainment as a feature. You must use a scaler after that.

   (a) First run a logistic regression with OVR and solver as 'lbfgs' with no penalty. What is the prediction accuracy of the train and test sets?

   (b) Try the l1 penalty by using `liblinear` solver and C value of 0.1 (the multi class will continue to be OVR). We want to see if it improves the test accuracy. *State what you observed as the train and the test accuracy. Be sure to identify the features that resulted in a (nearly) zero value for the β coefficient.*

   (c) Next try l2 penalty, everything else being the same as above in (b). Compare the accuracy of the train and test predictions (against the previous two cases).