Predicting Credit Card Approvals

Task 1: Instructions

Load and look at the dataset.

- Import the `pandas` library under the alias `pd`.
- Load the dataset, `"datasets/cc_approvals.data"`, into a `pandas` DataFrame called `cc_apps`. Set the `header` argument to `None`.
- Print the first 5 rows of `cc_apps` using the `head()` method.

Task 2: Instructions

Inspect the structure, numerical summary, and specific rows of the dataset.

- Extract the summary statistics of the data using the `describe()` method of `cc_apps`.
- Use the `info()` method of `cc_apps` to get more information about the DataFrame.
- Print the last 17 rows of `cc_apps` using the `tail()` method to display missing values.

Helpful links:

- pandas `tail()` method **documentation**

Task 3: Instructions

Split `cc_apps` into train and test sets.

- Import `train_test_split` from the `sklearn.model_selection` module.
- Drop features 11 and 13 using the `drop()` method.
- Using the `train_test_split()` method, split the data into train and test sets with a split ratio of 33% (`test_size` argument) and set the `random_state` argument to 42. Assign the train and test DataFrames to the following variables respectively: `cc_apps_train`, `cc_apps_test`.

Keep track of the total number of features before and after dropping the features. This often helps with debugging.

Setting `random_state` ensures the dataset is split with same sets of instances every time the code is run.

Helpful links:

- pandas `drop()` method **documentation**
- sklearn `train_test_split()` method **documentation**

Task 4: Instructions

Replace the question marks with NaN.

- Import the `numpy` library under the alias `np`.
- Replace the '?'s with NaNs using the `replace()` method in both the train and test sets.

Helpful links:

- pandas `replace()` method **documentation**
- NumPy data types for **special values**

Task 5: Instructions

Impute the NaN values with the mean imputation approach.

- For the numeric columns, impute the missing values (NaNs) with pandas method `fillna()`. Ensure the test set is imputed with the mean values computed from the training set.
- Verify if the `fillna()` method performed as expected by printing the total number of NaNs in each column.

Remember that you have already marked all the question marks as NaNs. pandas provides `fillna()` to help you impute missing values with different strategies, mean imputation being one of them. pandas also has a `mean()` method to calculate the mean of a DataFrame. As your dataset contains both numeric and non-numeric data, for this task you will only impute the missing values (NaNs) present in the columns having numeric data-types (columns 2, 7, 10 and 14).

Helpful links:

- mean imputation **tutorial**
- pandas `fillna()` method **documentation**
- pandas `mean()` method **documentation**
- pandas `isnull()` method **documentation**

Task 6: Instructions

Impute the missing values in the non-numeric columns.

- Iterate over each column of `cc_apps_train` using a `for` loop.
- Check if the data-type of the column is of `object` type by using the `dtypes` keyword.
- Using the `fillna()` method, impute the column's missing values with the most frequent value of that column with the `value_counts()` method and index attribute and assign it to both `cc_apps_train` and `cc_apps_test`. Ensure that the test DataFrame's (`cc_apps_test`) columns are imputed using data from the training DataFrames (`cc_apps_train`).
- Finally, verify if there are any more missing values in the DataFrames that are left to be imputed by printing the total number of NaNs in each column.

The column names of a pandas DataFrame can be accessed using `columns` attribute. The `dtypes` attribute provides the data type. In this part, `object` is the data type that you should be concerned about. The `value_counts()` method returns the frequency distribution of each value in the column, and the `index` attribute can then be used to get the most frequent value.

Task 7: Instructions

Convert the non-numeric values to numeric.

- Use the `get_dummies()` method on `cc_apps_train` and then use it on `cc_apps_test`. Store the outputs in `cc_apps_train` and `cc_apps_test` variables respectively.
- Use the `reindex()` method on `cc_apps_test` using the columns from `cc_apps_train`. Use 0 to fill the columns that aren't present in the reindexed `cc_apps_test`.

The last reindexing step is used for discarding any new categorical feature that'd appear in the test data.

Helpful links:

- Use `get_dummies()` method to deal with new values **[Stack Overflow answer]**
- pandas `get_dummies()` method **[documentation]**
- pandas `reindex()` method **[documentation]**

Task 8: Instructions

Rescale the features of the data.

- Import the `MinMaxScaler` class from the `sklearn.preprocessing` module.
- Segregate the features and labels into `X_train`, `y_train`, `X_test`, and `y_test` variables respectively. You can use `iloc` for this purpose.
- Instantiate `MinMaxScaler` class in a variable called `scaler` with the `feature_range` parameter set to `(0,1)`.
- Fit the `scaler` to `X_train` and transform the data, assigning the result to `rescaledX_train`.
- Use the `scaler` to transform `X_test`, assigning the result to `rescaledX_test`.

When a dataset has varying ranges as in this credit card approvals dataset, one a small change in a particular feature may not have a significant effect on the other feature, which can cause a lot of problems when predictive modeling.

Helpful links:

- Segregating the features and labels of a pandas DataFrame **Stack Overflow answer**
- sklearn's `MinMaxScaler` class **documentation**

Task 9: Instructions

Fit a `LogisticRegression` classifier with `rescaledX_train` and `y_train`.

- Import `LogisticRegression` from the `sklearn.linear_model` module.
- Instantiate `LogisticRegression` into a variable named `logreg` with default values.
- Fit `rescaledX_train` and `y_train` to `logreg` using the `fit()` method.

Helpful links:

- sklearn Logistic Regression **documentation**

Task 10: Instructions

Make predictions and evaluate performance.

- Import `confusion_matrix()` from `sklearn.metrics` module.
- Use `predict()` on `rescaledX_test` (which contains instances of the dataset that `logreg` has not seen until now) and store the predictions in a variable named `y_pred`.
- Print the accuracy score of `logreg` using the `score()`. Don't forget to pass `rescaledX_test` and `y_test` to the `score()` method.
- Call `confusion_matrix()` with `y_test` and `y_pred` to print the confusion matrix.

Helpful links:

- sklearn confusion matrix **documentation**

Task 11: Instructions

Define the grid of parameter values for which grid searching is to be performed.

- Import `GridSearchCV` from the `sklearn.model_selection` module.
- Define the grid of values for `tol` and `max_iter` parameters into `tol` and `max_iter` lists respectively.
- For `tol`, define the list with values 0.01, 0.001 and 0.0001. For `max_iter`, define the list with values 100, 150 and 200.
- Using the `dict()` method, create a dictionary where `tol` and `max_iter` are keys, and the lists of their values are the corresponding values. Name this dictionary as `param_grid`.

Grid search can be very exhaustive if the model is very complex and the dataset is extremely large. Luckily, that is not the case for this project.

Task 12: Instructions

Find the best score and best parameters for the model using grid search.

- Instantiate `GridSearchCV()` with the attributes set as `estimator = logreg`, `param_grid = param_grid` and `cv = 5` and store this instance in `grid_model` variable.
- Fit `rescaledX_train` and `y_train` to `grid_model` and store the results in `grid_model_result`.
- Call the `best_score_` and `best_params_` attributes on the `grid_model_result` variable, then print both.
- Extract the best model from `grid_model_result` and evaluate it on the test set (`rescaledX_test`, `y_test`).

Grid searching is a process of finding an optimal set of values for the parameters of a certain machine learning model. This is often known as hyperparameter optimization which is an active area of research. Note that, here we have used the word parameters and hyperparameters interchangeably, but they are not exactly the same.