Database Information Searcher

Content

Part one: Background

Because of the expansion of the company, my leadership needs to integrate all the information of the original/old company database into the new database. At the same time, valuable data content needs to be collected from a financial information platform. (There is such a financial data sharing platform, where many financial companies share business data. Any member company can collect data information according to its own needs. Roughly estimated, the total number of databases exceeds 100,000).

At the same time, the database information in the platform has the following characteristics:

1) Database table names and attribute names that store the same type of data are similar. Usually, they are expressed in fixed technical terms.

2) However, there are significant differences in the way table names and attribute names are written in different databases. The main noises are of the following categories: redundant separators, insignificant numbers or character suffixes, full name and abbreviated expressions.

3) A few of databases in the platform, the information of the table name and attribute name should be mapped to multiple table names and attribute names in the new database

4) Table names and attribute names that store different information are similar. Sometimes this phenomenon is common. For non-financial professionals, some table names, attribute names look the same, and there is not much difference.

5) Dataset: 00_GoBD_structure_complete.xlsx

Project Expectation

1) Develop an algorithm that immediately recognizes whether the input table name and attribute name are valuable for Auren. If it is valuable, the table and column information should map to the which table and column in the new database.

2) Every valuable database information should to be detected. The accuracy rate should trend to 100%.

3) If the information of the table name and attribute name should be mapped to multiple table names and attribute names in the new database. Then output all of new database table names and attribute names.

4) The basic principles of the algorithm should be easy to understand, any software engineer can understand and extend it, even he/she is not a data engineer.

5) The code is simple and clear. In the future expansion programming, bugs are easy to find and solve.

6) The results of multiple judgments are output for each table name and attribute name, and are arranged according to the size of the possibilities. At the same time, the number of output judgment results can also be freely changed.

7) The '00_GoBD_structure_complete.xlsx' show mapping relationship between old database and new database and which table/column is important.

Part two: analysis

1) Because of the input (table, column name) is unknown when you use the algorithm. So it should have an ability to remove any type of noise.

2) This is a classify issue. We should consider supervised learning algorithm.

3) We should convert string(name) information to number, at the same time, descript this string meaning by the number.

4) This model should be a widely valid algorithm for current needs.

5) After converting to the digital realm, how to make the distance between similar semantic names very close, and the distance between different semantic names is far enough.

Part three: Candidate solution

1) Word2vec:

This algorithm can convert name into number. And it is easy to training.

The programming logic is: by trained word2vec model, we can calculate the distance between input name and labeled name(table or column name). The closer the distance, the higher the similarity between the names, the more likely it is the result we want.

But, it is a bad solution. Because, the noise issue. We can't find out a general method to remove all of noise in name. At the same time, There is many financial terms is unique. The training corpus is not existed.

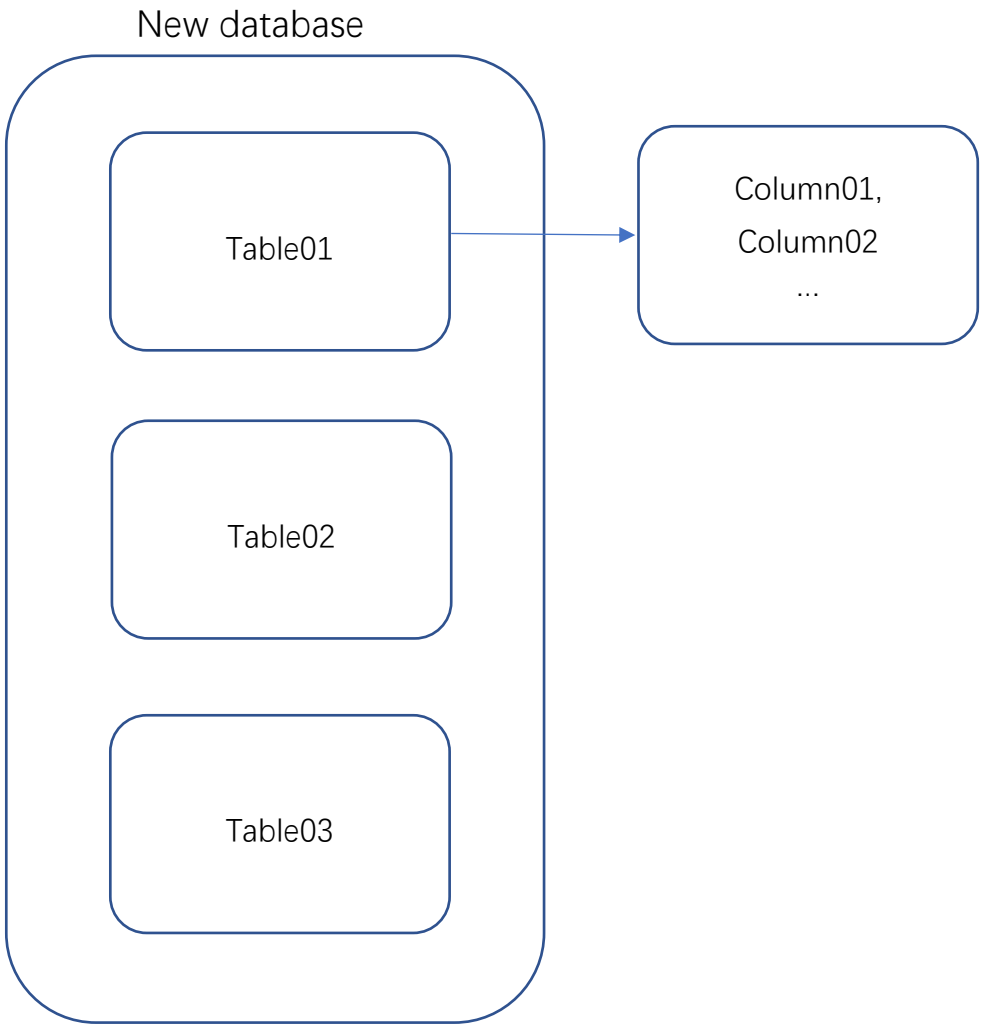
2) Word2number

This algorithm can convert name into vector by a specific principle. And you just need to select a good principle. Then the problem solved.

This algorithm not to consider the meaning of the name. We consider the structure of the name.

For each table/column name, we just need to get unique sub-string. By all of the unique sub-string we can create a good principle to convert name to string.

When we use the model to predict some name, we also need to convert the input string by specific principle. (For different table, the principle of column name is different.)



For all of table name, it has a convert principle. In each table, there is a unique convert principle for columns.

Part four: Programming

The structure of the source code has three part:

1) 'predict_main.py' is main function;

Use: You can call it in command line when you want to use it.

The programming **logic** is:

class ForeTeller:

→ '__init__' method: define the table principle and columns principle. (it should update when you training model by new dataset)

→ 'predict_table' method: use trained model to predict input table name, output predicted result with possibility. (Output order, arranged by possible values)

→ 'predict_table_top_N' method: based on user setting. Select top N table as output.

→ 'predict_column_improve' method: based on table predict result, use specific column principle to predict column values. Select top N column for each table as output.

→ 'start' method: call method to predict table, column and important value/number. This important number(0-1) indicated whether this input is valuable. (0 is no valuable, 1 is important).

class DataFlow:

→ 'read_data_commandline' method: read information

2) xx_xxxx_classifier:

That is for training table, columns predict model. There are three steps: extract information from dataset, training model and cross Validation.

The size of dataset not enough to cover all of mapping feature(new table, columns) if we split it as training set and test set. Some mapping relation will loss. So, we enlarge the dataset 100 times. (more or less) Then split it.

3) importance_evaluation:

The code in this file is used to create a model to calculate the importance of the input value. We use same trick to training column model: word2number. For table model: by KNN algorithm.

For this issue, only part of table has valuable column. And others are no value.

Turn to folder: 'importance_evaluation': There are three .py file.

1) '00_table_evaluate.py': training table model by KNN. Because of no label available to indicated if this table is important.

2) '00_column_important_value_evaluate.py', it is a model factory for create models to detect if the column name is important.

3) 'aa_importance_evaluate_plus.py', it has the main function we need to call, when make an important prediction for input value.

Part five: Future extension

When user has new mapping relationship need to update. You just need analysis all new table name feature, extract keywords. (You can programming to select or select it by yourself.).

By this new principle, you can training new table predictor. For column model, the new principle created by source code is better idea.

For important evaluation model, the idea is same

Remember, you should use same word2number principle when training table or column mapping relationship and using these models. This is commonly bug.

If you want to change the output number of table prediction. Just setting value in 'predict_main.py'. line 349.

If you want to change the output number of column prediction. Just setting value in 'predict_main.py'. line 339.

For user and model developer, the format of input string both unpredictable in future. So, I believe this principle is robust and easy to upgrade.

In the future, if the new database has similar table names, we also can easily solve it by zoom in the difference between them. (how to zoom in: repeat the difference substring in principle.)

Eg: predict_main.py, line 13, self.list_table_vector_criterion_new