University of Nebraska - Lincoln
DigitalCommons@University of Nebraska - Lincoln

Computer Science and Engineering: Theses, Dissertations, and Student Research

Computer Science and Engineering, Department of

7-27-2012

Data Mining of Protein Databases

Christopher Assi
University of Nebraska-Lincoln, cjassi08@yahoo.com

Follow this and additional works at: http://digitalcommons.unl.edu/computerscidiss

Part of the Computer Engineering Commons, Databases and Information Systems Commons, and the Organic Chemistry Commons

Assi, Christopher, "Data Mining of Protein Databases" (2012). *Computer Science and Engineering: Theses, Dissertations, and Student Research*. 46.
http://digitalcommons.unl.edu/computerscidiss/46

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in Computer Science and Engineering: Theses, Dissertations, and Student Research by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

DATA MINING OF PROTEIN DATABASES

by

Christopher Assi

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Peter Revesz

Lincoln, Nebraska

August, 2012

DATA MINING OF PROTEIN DATABASES

Christopher Assi, M. S.

University of Nebraska, 2012

Adviser: Peter Revesz

Data mining of protein databases poses special challenges because many protein databases are non-relational whereas most data mining and machine learning algorithms assume the input data to be a relational database. Protein databases are non-relational mainly because they often contain set data types. We developed new data mining algorithms that can restructure non-relational protein databases so that they become relational and amenable for various data mining and machine learning tools. We applied the new restructuring algorithms to a pancreatic protein database. After the restructuring, we also applied two classification methods, such as decision tree and SVM classifiers and compared their accuracy in predicting whether particular pancreatic proteins are involved in pancreatic cancer. From our prediction the SVM gave us not only the highest accuracy, about 73%, but it also gave the most consistency among the GO terms and PFAM family proteins.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Peter Revesz for his optimism, guidance and feedback during the course of my research and the writing of the thesis. He always stood by my side during challenging times. I'm grateful for having Dr. Jitender Deogun and Dr. Anita Sarma as part of my examination committee. I have benefited from taking several excellent classes from them during my years at the University of Nebraska-Lincoln.

I would also like to thank Dr. Robert Powers and his Ph.D. student Bradley Worley from the Department of Chemistry at the University of Nebraska-Lincoln for providing me the pancreatic cancer-related database. They also helped to motivate the work by explaining to me many aspects of protein and pancreatic cancer.

Last but not least, I would like to give a special thanks to my undergrad advisor Dr. Lalchand Shimpi from Saint Augustine's University for his support of my thesis and also thanks my family and friends for standing behind me during the tough times and always giving me kind words of encouragements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"The Human Protein Reference Database represents a centralized platform to visually depict and integrate information pertaining to domain architecture, post-translational modifications, interaction networks and disease association for each protein in the human proteome."

Prasad, 2009[16]

Data mining is a greatly successful and expanding field that combines statistical analysis, machine learning and database technology to extract hidden patterns and relationships from large databases [24]. Fayyad [7] defines data mining as "a process of nontrivial extraction of implicit, previously unknown and potentially useful information from the data stored in a database." Giudici [8] defines it as "a process of selection, exploration and modeling of large quantities of data to discover regularities or relations that are at first unknown with the aim of obtaining clear and useful results for the owner of database."

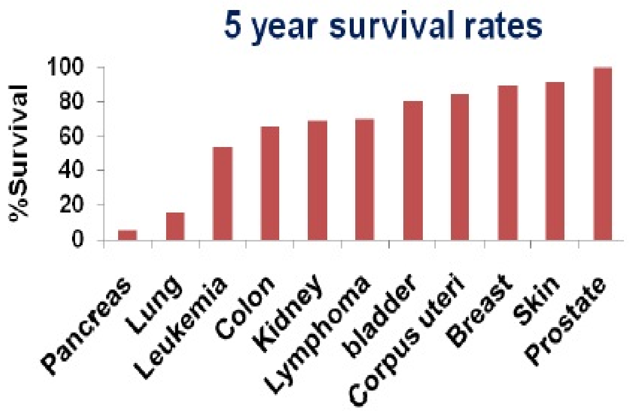


Figure 1.1: Cancer Survival Rates

Data mining is increasing applied to novel and non-traditional types of databases. Our primary focus in this thesis is protein databases. We work on a particular pancreatic cancer-related protein database on an interdisciplinary project in collaboration with Dr. Robert Powers and Bradley Worley in the Department of Chemistry at the University of Nebraska-Lincoln [25]. In particular, they have contributed the database used in the thesis. Their database was a collection of several related databases that were described in earlier papers regarding pancreatic cancer [2,3,4,9,12,22,29]. However, the thesis contains ideas that are generally applicable to other protein databases and in fact to other databases that have a similar structure.

Protein databases are very important because they give us a starting point on doing research. By having the databases, we are able to study the proteins in depth and understand where the needs of change have to be. Without the protein databases, we wont have the knowledge on which proteins to focus our work in. There are thousands of pancreatic proteins and grouping them together is a focal key on our research.

Pancreatic cancer proteins have the lowest survival rate among cancer. The survival rate is barely over 5% chance [15] as showed in figure 1.1. In consequence, the need of early prediction of pancreatic cancer is greatly needed. We need to know whether anomalous proteins, which is the unusually high or low levels in a patient predicts future pancreatic cancer. If the anomalous protein is cancer-related, it may be blocked or reduced by some drug. In a nutshell, preventing the development of cancer is what our research long-term goal is intending to do.

Our pancreatic protein databases, which were obtained from the Department of Chemistry, had some limitations. Dr. Robert Powers and Bradley Worley designed the databases in a way that we could not run certain types of SQL queries. Therefore, we reformatted the databases in a way that we could run new queries and data mining software. Our research shows several ways we could overcome those challenges by describing effecting way to change the structure of the databases. Our new structure databases are not only feasible for computer scientists but also will help biologists continue doing their work in a more defined way. They would not be running in issues such as having multiple many to many attributes throughout the databases. They will now be able to see a much clear databases with easy understanding throughout.

Data mining of protein databases poses special challenges because many protein databases are non-relational whereas most data mining and machine learning algorithms assume the input data to be a relational database. Protein databases are non-relational mainly because they often contain set data types. This thesis was motivated by the need to solve the special problems posed by the non-relational nature of the protein databases that we were asked to consider for data mining. Solving the data mining problems was an important part of the joint work, and we focus on that aspect in this thesis, although we contributed to solving other database issues related to the joint project.

1.1 Summary of Contributions

The three major contributions of the thesis can be listed as follows:

1. **Restructuring algorithms:** The goal of our research was to define a well structure way to better understand the pancreatic protein databases. The databases were designed in a way not practicable for non-biologists. During our preliminary analysis while we were studying and working on the databases, we ran into issues where the databases had large amount of redundancies throughout. We were conflicted on which approach to take. Moreover, because our focus was to use data mining techniques, it was inevitable we come to a solution on how to solve the problem. This problem did not bother the Biologists community because it was never such an issue ever been reported. Our solution was to come up with high-level SQL queries, which will help solve the problem. During our initial phases, all our SQL queries failed. We were unable to recognize our problems at the start of the research project. It was not until we chose to move the two databases into one that we were able to located our problem, which was because we were dealing with two pancreatic protein databases on two different databases. Our goal was then to bring the two databases together into one single database. We then proceed it with that approach by creating a new database and moving the pancreatic cancer proteins from the two databases into one.
   In addition, we developed algorithms to restructure non-relational protein databases so that they become relational and amenable for various data mining and machine learning tools. The restructuring algorithms were partly based on the MySQL databases system and written in part in the Perl programming language. As examples, we restructured several non-relational tables from a pan-

creatic cancer-related protein database. This protein database included several large tables, such as, the pancreatic protein GO term and PFAM family classification tables. GO terms and PFAM family groups describe characteristics of proteins. The restructuring method and the details of the restructuring of these tables are both described in Chapter 3.

2. **WEKA data formatting and extension:** We also formatted our data as an "ARFF" file, which is the usual input data format to use in WEKA. The ARFF format which stands for Attribute Relation File Format, has a specific structure we had to follow to put our data in. Below is a general example to how the format should look:

   The dataset has to start with a declaration of its name

   @relation name

   The declaration of the relation name is then followed by a list of all the attributes in the dataset including the class attribute. These declarations have the form

   @attribute attribute_name specification

   If an attribute is nominal as was our data, specification contains a list of the possible attribute values in curly brackets. For our data the attributes, which were in the curly bracket, were all the UIDs of the database:

   @attribute nominal_attribute {first_value, second_value, third_value}

   In addition to these two types of attributes, there also exists a string attribute type. This attribute provides the possibility to store a comment or ID field for each of the instances in a dataset. In our case the attribute name we chose was called relation were we designated either a '0' or '1' to differentiate between the pancreatic and non-pancreatic protein database:

   @attribute relation { 0, 1}

After the attribute declarations, the actual data is introduced by a

@data

The data is where we listed of all our instances. The instances were our newly restructured data. Our process started with a raw data which had a lot of redundancies, was then transformed by eliminating all redundancies into a new format, which could be used by computer scientist as well as non-scientists. By the time the restructured data was completed it was then converted in comma-separated values (csv) format that was added to the data section of the ARFF file.

The comma-separated values format is frequently used to transfer large amount of data between databases or applications, which are not directly join. The records are editable using Microsoft Excel spreadsheets where the fields are separated by commas [6].

Moreover, we have added to the WEKA the libSVM (Library for Support Vector Machines) software implementation of support vector machines. This addition made the WEKA package more useful for our project and made the project more efficient because the other data mining algorithms we used were all available in WEKA already. Hence all these data mining algorithms, including libSVM could be run on the same input data without any additional effort. We describe the WEKA system and this addition to the WEKA system in Chapter 2

3. **Experiments:** WEKA is a java program that contains a large variety of tools that can be used for pre-processing datasets, so that we can focus on our algorithm without considering too much details as reading the data from files, implementing filtering algorithm and providing code to evaluate the results. There are several ways to learn machine learning algorithms and several plat-

forms available, but to be able to combine all the platforms into one is a major bonus. WEKA gives us that flexibility. This tool allows us to classify and predict the outcome of pancreatic cancer.

Moreover, after the restructuring the pancreatic cancer database and extending by libSVM using the WEKA system, we applied several classification methods. In particular, we describe the results of our experiments with the J48 decision tree and the libSVM support vector machine.

SVM is an effective technique for data classification. Support vectors are the data points that lie closest to the decision surface, also called the hyperplane [7]. We do not expect everyone to have a great understanding of the underlining hypothesis behind SVM. We introduce the main objective of our research by using this technique to explain our procedures. The libSVM, which refers to the Library of Support Vector Machine, is an integrated software, which has a better optimization to the Support Vector Machine (SVM) because it runs much faster and the efficiency has been enhanced.

The decision tree is referred as the J48 when using WEKA. It is defined as a tree whose internal nodes are tests and whose leaf nodes are categories. In a nutshell, the non-leaf nodes are labeled with attributes and the leaves of the tree are labeled with classications. The decision tree is a widely popular algorithmic method when dealing with data mining and classification in general.

The purpose of using multiple classifiers was to compare their accuracy and identify the most efficient among them. From our results, the SVM holds an edge over the decision tree. Our detailed test results are presented in Chapter 4.

1.2 Outline of the Thesis

This thesis is structured as follows. Chapter 2 describes some basic background that is needed to understand the work in the thesis. In particular, Chapter 2 gives a brief description of classifiers and the WEKA system. Chapter 3 presents the restructuring method and illustrates it on sample protein databases. Chapter 4 gives the results of applying the J48 decision tree and the libSVM classifiers to the restructured pancreatic cancer database. Chapter 5 gives our conclusion and possible directions for future work.

Chapter 2

Background Concepts and Tools

This chapter provides a background to the main concepts and tools used in the thesis. Section 2.1 gives an introduction to classifiers and Section 2.2 describes the WEKA system that contains a library of implemented classifiers.

2.1 Classifiers

There are problems in which we need to classify items in many instances, that is, we need to predict some distinctive of an item based on several of its parameters. A variable is represented by every parameter, which can then also a take a numerical value. The variable is called a *feature* and the set of variables is called the *feature space*. The dimension of the feature spaces is the number of feature.

The concrete feature of the item we want to predict is called the label or class of the item. We use the classifier to make our predictions. A feature space X to a set Y labels maps each classifier. The classifiers are built using machine learning algorithms, which are able to automatically improve by the analysis of data sets, a rationale reason we called them machine learning because they learn by experience

[24].

Let R(x_1,...,x_n, y) be a relation, where the set of attributes X = {x_1,...,x_n} is called the *feature space* and the y attribute is called a *label.* Each tuple of the relation describes some entity based on specific values of the feature space and the label. For example, each row may describe a protein with specific feature attributes, such as, the proteins molecular weight, its amino acid sequence etc. and a label attribute, such as, whether it is involved in pancreatic cancer.

Given such a relation R, a classifier is mapping from X to y. If a classifier is correct on all tuples of relation R, then the value of y can be always predicted from the values of X. In practice, the classifier may not be correct on all proteins. Further, classifiers are intended to be able to classify even those proteins that are new, not just those that are already in R. That is a difficult task and perfection is not to be expected. However, a classifier that has a fairly high accuracy could be still useful in many applications.

For example, suppose that a physician detects that a patient has an unusually high presence of a new protein. In this case, a classifier may predict that the new protein is involved in cancer based only the feature space. Then the doctor may prescribe some drug that either reduces the amount of the protein in the patients cells or blocks the action of the protein in the cells. Such a treatment could prevent the formation or spread of cancer.

There are many classifiers. *Decision trees* are popular classifiers. A decision tree is a tree which is read from the root towards the leaves, and whose internal nodes are tests and whose leaf nodes are categories [28]. For example, C4.5 is a well-known decision tree algorithm [18].

A *Support Vector Machine* (SVM) performs classification by constructing for relation R an n-dimensional hyperplane that optimally separates the data into two categories (for example when y=0 and y=1) [14]. An example of SVM is the libSVM implementation.

There are also various approaches to determine the performance of classifiers. The performance can most simply be measured by counting the proportion of correctly predicted examples in an unseen test dataset. This value is the accuracy. The simplest case is using a training set and a test set which are mutually independent. This is referred to as holdout estimate. To estimate variance in these performance estimates, holdout estimates may be computed by repeatedly resampling the same dataset. For example, randomly reordering it and then splitting it into training and test sets with a specific proportion of the examples, collecting all estimates on test data and computing average and standard deviation of accuracy.

A more elaborate method is cross-validation[5]. Here, a number of folds n is specified. The dataset is randomly reordered and then split into n folds of equal size. In each iteration, one fold is used for testing and the other n-1 folds are used for training the classifier. The test results are collected and averaged over all folds. This gives the cross-validation estimate of the accuracy. The folds can be purely random or slightly modified to create the same class distributions in each fold as in the complete dataset. In the latter case the cross-validation is called *stratified.* . An n-fold cross-validation is called stratified when the folds are selected so that the mean response value is approximately equal in all the folds. In the case the classification is dichotomous, this means that each fold contains two types of the same proportions class labels [19].

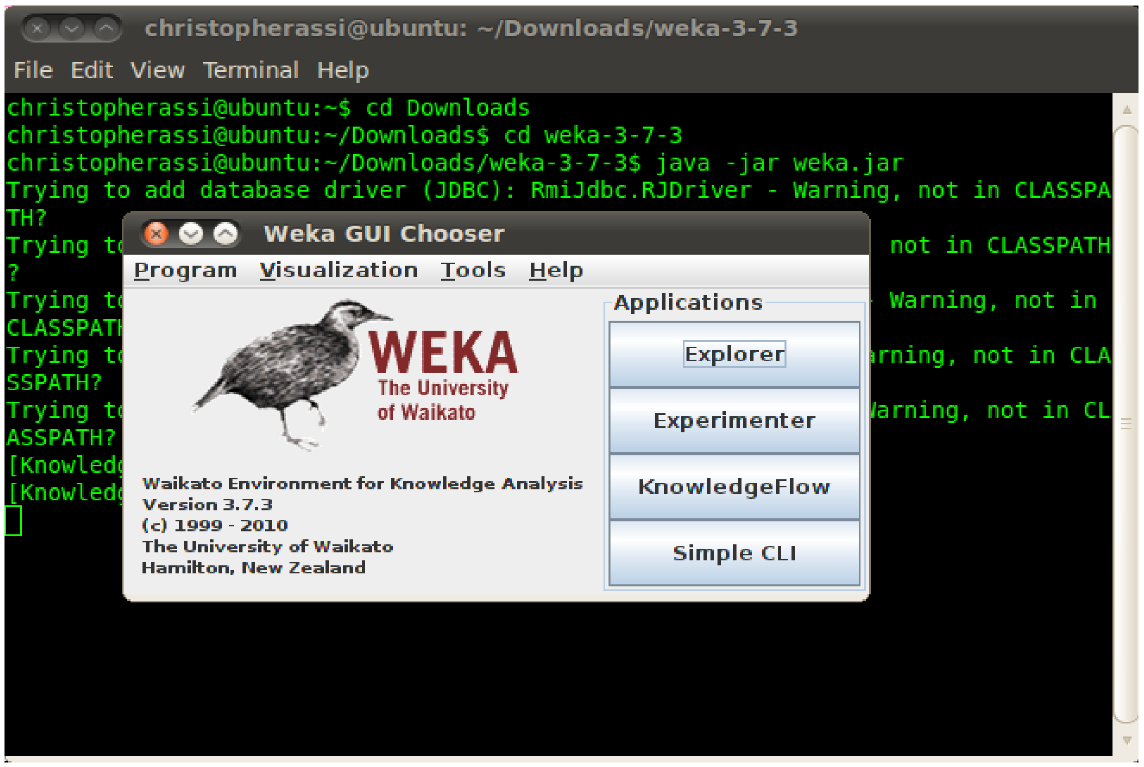
2.2 The WEKA Library

"The programme aims to build a state-of-the-art facility for developing techniques of machine learning and investigating their application in key areas of the New Zealand

economy. Specifically we will create a work-bench for machine learning, determine the factors that contribute towards its successful application in the agricultural industries, and develop new methods of machine learning and ways of assessing their effectiveness."

Hall [10]

Data mining and classification were a main objective to our research but without the tool to run the experimentation with, they would be unnecessary. The tool we used is called WEKA. The Waikato Environment for Knowledge Analysis (WEKA) system was developed at the University of Waikato, New Zealand [26]. WEKA is an extensive library of data mining and machine learning algorithms.


christopherassi@ubuntu: ~/Downloads/weka-3-7-3
File Edit View Terminal Help
Weka GUI Chooser
Program Visualization Tools Help
Applications
Explorer
Experimenter
KnowledgeFlow
Simple CLI
WEKA
The University of Waikato

```
christopherassi@ubuntu:~$ cd Downloads
christopherassi@ubuntu:~/Downloads$ cd weka-3-7-3
christopherassi@ubuntu:~/Downloads/weka-3-7-3$ java -jar weka.jar
Trying to add database driver (JDBC): RmiJdbc.RJDriver - Warning, not in CLASSPA
TH?
Trying to                                                  not in CLASSPATH
?
Trying to                                                  Warning, not in
CLASSPATH
Trying to                                                  arning, not in CLA
SSPATH?
Trying to                                                  Warning, not in CL
ASSPATH?
[Knowledg
[Knowledg
```

Waikato Environment for Knowledge Analysis
Version 3.7.3
(c) 1999 - 2010
The University of Waikato
Hamilton, New Zealand

Figure 2.1: WEKA GUI Chooser

In WEKA, the input data is a relation or table which is represented by an Attributes Relation File Format (ARFF) file. Each ARFF file starts with a title to let the user know what kind of data is stored in the file. The title is followed by a relation type and the all the attributes and their types. Each attribute can have one of the following data types:

- *nominal*, a user-defined set of values,
- *numeric*, a real number,
- *string*, an arbitrary long list of characters, or date.

Finally, the attribute declarations are followed by the actual data rows. Once the WEKA graphical user interface is started, as shown in Figure 2.1, one can use the Explorer button to enter the system and then load the ARFF file. After that one can select from a menu of library options any particular classifier, including the J48 decision tree (a close implementation of the C4.5 decision tree) or the libSVM support vector machine implementation, which is not a basic option, but we also added to our own copy of the WEKA library.

Chapter 3

The Restructuring Method

The Borg collection of databases contains two databases Borg_pdac, which contains data about pancreatic proteins that are related to pancreatic cancer and Borg_np, which contains data about pancreatic proteins that are not involved in cancer. The two different databases needed to be merged into one database. Section 3.1 describes some of the challenges we have overcame. Section 3.2 describes the process of merging two databases Borg_pdac and Borg_np and Section 3.3 describes the restructuring of the merged database so that it is ready for input to various data mining algorithms. Finally, section 3.4 describes the process of merging the GO_merge and the PFAM_merge together.

3.1 Challenges to Overcome

Restructuring the data was a problem we have encountered, but it was definitely not the only issue we faced. Having to work on two different databases was an issue as well as having to come up with high-level SQL queries to merge the two databases. Our issues were new in the database communities because they were never a need for them to having to change the format of a database. We spent countless time experimenting different SQL queries, which would work for our dilemma. Once we came up with some good results, they were practically impossible to use them because it was unlikely for a user to have to write 200 to 300 lines or more of SQL queries.

In addition, we could not afford to make any typing mistakes, as it has never been a way to go back and fix any mistakes while in the SQL query command. That was a huge drawback we had to overcome while using SQL queries. The solution to it was to write a program as well as the SQL queries to help us automatically generate those SQL queries lines we have constructed. By doing so, we spared ourselves the part of a typing any errors as well as the time consuming to write the queries.

Both the Borg_np and the Borg_pdac database have the same structure tables, in which their tables contain the same set of attributes. Among the tables that are of interest to us are the GO and the PFAM tables. Both Borg_np and Borg_pdac contain 40 different tables. Out of those 40 tables, we used the following 4 tables, which had a total number of 125,545 rows.

GO_np	$70,331$ rows
GO_pdac	$30,888$ rows
PFAM_np	$7,054$ rows
PFAM_pdac	$7,272$ rows

Table 3.1 displays the UID on the left and GO terms on the right. We encountered huge problems involving many-to-many relations in the GO table. For example, we can see that rows three and five with the same UID O43491 are related to two different GO terms, GO:0005886 and GO:0019898. On the other hand, similarly, rows three and eight with the same GO term GO:0005886 are related to two different UIDs,

O43491 and Q96C24. Restructuring the tables to make the UIDs primary keys and unique in each row was one of the challenges we faced during our work.

The GO table lists all (UID, GO) pairs, such that UID is the universal identifier of a pancreatic protein and GO is a feature descriptor, also called a GO term. Each UID can be associated with a set of GO terms and each GO term can be associated with a set of UIDs. A simplified version of the GO table in Borg_pdac looks as follows:

UID	**GO**
O43491	GO:0003779
O43491	GO:0005198
O43491	**GO:0005886**
O43491	GO:0008091
O43491	**GO:0019898**
O43491	GO:0030866
Q96C24	GO:0005215
Q96C24	**GO:0005886**
Q96C24	**GO:0019898**
Q96C24	GO:0030658
Q96C24	GO:0042043

Table 3.1: GO table

The PFAM table is similar to the GO table. The PFAM table contains the UID of proteins and the PFAM terms. PFAM terms form another set of characterizations of proteins that is sometimes a useful alternative to the GO term characterization.

Before we can do any data mining, we have to merge each pair of corresponding tables from Borg_np and Borg_pdac. We illustrate this process only for the two GO tables because the merging of the two PFAM tables and other pairs of corresponding tables is similar.

Some of the challenges we faced at the beginning were on how to work on the tables knowing they were on two different databases. The analysis we wanted to do consisted of us having the two tables in the same database, so we could run our queries. We spent countless hours contemplating different scenarios as on how to go about the challenge of bringing the two databases together into one big database. The steps on how we solved this problem are described in Section 3.2.

Another challenge we faced was the process of flattening our data. Flattening is the process of transforming a data with identical records to be re-ordered in the same columns or rows so that way we eliminate repeated records. The idea of flattening was first brought to my attention during one our meetings at the Chemistry Department. The idea was on how to bring all the raw data into a new file, which not only eliminates redundancy but then also applies a way to show the relationship among the UIDs. To find a way to flatten our data was critical because data mining works well only when the database relation has a flattened structure.

Furthermore, the cancer database we have been working with does not use any ordering to arrange the proteins. The ordering for the data is depending just on which proteins is identified first. In our case for the purpose of data mining it is important to use some sort of ordering method. Some tables are more complex than others to come up with an ordering process. For example, during the course of this thesis we have used the GO and PFAM tables to generate our analysis. The GO and PFAM went through various transformations in order to be used in our data mining technique. In addition, we have many to many relations tables in our data. They are tables in which same UIDs have many different GO term or a single GO term has an UID which has many other GO terms related to. The solution of these challenges, that is, how the process of flattening works is described in Section 3.3.

Furthermore, we wanted to expend our work to also merge our flattening data. Our goal was primary to merge the GO_np with the GO_pdac and as well as the PFAM_np with PFAM_pdac because they were on different database so we could better run our analysis. Once merged, the GO_np and the GO_pdac were called GO_merge and the PFAM_np and PFAM_pdac were called PFAM_merge. Our next goal was then to come up with an even higher SQL query to merge the two already merged data. Besides dealing with SQL query complexity, the issue we were having on merging of the two tables (GO_merge and Pfam_merge) dealt with ambiguity. Ambiguity in database occurs when more than one table is used in a 'JOIN', there may be two columns with the same name. In our case, the GO_merge table has two of the same column names as Pfam_merge table, which are 'UID' and 'Y'. Our challenge was on to solved two ambiguity columns while joining the two merged tables.

3.2 Merging Cancer and Non-Cancer Data

In order to merge the two GO tables, we exported the GO tables from both Borg_np and Borg_pdac. First, we export the GO table from the Borg_np database into a comma separated values (.csv) file with the following command:

```
mysql> select UID, GO
    – > from GO
    – > into outfile '/tmp/GO_np.csv'
    – > fields terminated by ','
    – >optionally enclosed by '"'
    – >lines terminated by \n;
```

Similarly, we export the GO table from the Borg_pdac database into a .csv file as follow:

```
mysql> select UID, GO
    − > from GO
    − > order by UID
    − > into outfile '/tmp/GO_pdac.csv'
    − > fields terminated by ','
    − > optionally enclosed by '"'
    − > lines terminated by \n;
```

Then we created a new database called Merge using the following query:

```
mysql> create database Merge
```

The next phase is to create two new tables in our Merge database in which we will then import the two tables we have exported from the Borg database into. Two new tables named GO_np and GO_pdac were created using SQL. The first query created the GO_np table:

```
mysql> create table GO_np (UID varchar(16), GO varchar(16));
```

The second query created the GO_pdac table:

```
mysql> create table GO_pdac (UID varchar(16), GO varchar(16));
```

SQL query always check whether the input values are numerical or strings therefore it is very important to specify those type of values while creating tables. In our case our value type were varchar which are a mixture of variables and characters. Our next step is to import the two previously exported tables, which were the GO_np and the GO_pdac. To accomplish that task, we execute the following two queries:

```
mysql> load data local infile '/tmp/Flat/GO_np.csv'
    – > into table GO_np
    – >fields terminated by ','
    – > lines terminated by \n
    – > (UID, GO);

mysql> load data local infile '/tmp/Flat/GO_pdac.csv'
    – >into table GO_pdac
    – >fields terminated by ','
    – >lines terminated by \n
    – >(UID, GO);
```

Finally, we will be able to merge the GO_np and GO_pdac tables because now they are in the same Merge database. However, a simple union would lose the information whether the protein is related to cancer or not. Hence before the union, we extended the GO_np and the GO_pdac tables with a Y column, which denotes whether the protein is related to pancreatic cancer or not. All the proteins in the GO_np table will be extended with a Y value of "0", while all the proteins in the GO_pdac table will be extended with a Y value of "1". Below is the complete query:

```
mysql> create view GO_merge (UID, GO, Y) as
    – >select UID, GO, 0 from GO_np
    – >union
```

– >select UID, GO, 1 from GO_pdac;

After the above query is executed the GO_merge table looks as follows, where $\{\cdots\}$ indicates continuation of the table:

UID	**GO**	**Y**
O43491	GO:0003779	1
O43491	GO:0005198	1
O43491	GO:0005886	1
O43491	GO:0008091	1
O43491	GO:0019898	1
O43491	GO:0030866	1
Q96C24	GO:0005215	1
Q96C24	GO:0005886	1
Q96C24	GO:0019898	1
...	...	...

Table 3.2: GO_merge table

3.3 Data Restructuring

The next step in the process is a restructuring of the database. In this restructuring, which we also call *"flattening"*, the table is transformed into another table that has the same information but all information about a single protein appears in one row. For example, the above GO_merge table can be restructured or *flattened* as follows:

UID	3779	5198	5215	**5886**	8091	**19898**	30658	30866	42043	Y
O43491	1	1	0	**1**	1	**1**	0	1	0	1
Q96C24	0	0	1	**1**	0	**1**	1	0	1	1

Table 3.3: GO_merge_flat table

Table 3.3 has been transformed into a table that has the same information form table 3.2, but now all information about a single protein appears in one row. This table makes it lot easier to identify any redundancy throughout the database, which subsequently help us avoid them.

As it can be seen the number of attributes in the restructured relation is n+2, where n is the number of distinct GO terms. Apart from UID and Y, these distinct GO terms form the attributes of the restructured relation. Below each GO term a 1 or 0 indicates whether the GO term applied to the protein indicated by the UID on the left. For example, the GO term 3779 applied to UID O43491; therefore a 1 appears in row O43491 and column 3779.

The above is an explanation of restructuring. Next we describe the steps that we followed to get the actual restructured table. First, as a practical observation, we cannot actually restructure the entire GO_merge table because there are 7935 GO terms. Moreover, most of these GO terms occur very infrequently. Hence we selected only the top 200 most frequent GO terms as follows. First we found the frequency of each Go terms:

```
mysql> create view GOcount(GO,count) as
    − > select GO, count(*)
    − > from GO_merge
    − > group by GO;
```

The new table GOcount(GO,count) contains the count of each GO term. Once we have created the GOcount, we then have to extract the top 200 most frequent GO terms into a text file as follows:

```
mysql> select GO from GOcount
    −> order by count desc limit 200
    −> into outfile '/tmp/MergeTop200GO.txt';
```

Next we wrote a C++ program to help us automatically generate the SQL queries we have constructed to help us perform the restructuring. The program reads each line from the input file in ifstream ifs (name.txt) and output each line read into a SQL query format. The new format is then created into a new text file ofstream a_file (name.txt). The output file then can be used in a SQL query console to be run. The program is ran inside the Netbeans IDE software program. The main purpose of the program is to read the content from a file and add the written query from the program to it. The program will go over each line of the file, which is either the GO terms or PFAM pancreatic cancer proteins and add a query line until the end of the line. The program and its full structure is display in Appendix A of the thesis.

Moreover, the program while loop will be executed 200 times because the input file has 200 lines, each containing a single GO term. The program outputs the SQL query into a new text file called SQL_flatten.txt. Below is how the SQL_flatten.txt file looks like, where the {· · ·} means omitted lines. The complete output is displayed in Appendix B.

```
select UID,
max(case when GO = 'GO:0016021' then 1 else 0 end) as 'GO:0016021',
max(case when GO = 'GO:0005515' then 1 else 0 end) as 'GO:0005515',
max(case when GO = 'GO:0005634' then 1 else 0 end) as 'GO:0005634',
max(case when GO = 'GO:0005737' then 1 else 0 end) as 'GO:0005737',
max(case when GO = 'GO:0008270' then 1 else 0 end) as 'GO:0008270',
max(case when GO = 'GO:0006350' then 1 else 0 end) as 'GO:0006350',
max(case when GO = 'GO:0007165' then 1 else 0 end) as 'GO:0007165',
max(case when GO = 'GO:0005886' then 1 else 0 end) as 'GO:0005886',
max(case when GO = 'GO:0005524' then 1 else 0 end) as 'GO:0005524',
max(case when GO = 'GO:0003677' then 1 else 0 end) as 'GO:0003677',

.
.
.
Y
from GO_merge
group by UID
```

When the SQL query is executed, for each UID it checks all the GO terms. If any of the GO terms the UID is associated with matches a particular GO term for we are creating a column in the flattened table, then that GO term will get a value of "1" else it will get a value of "0". The process then continues until it does not read any more UID groups. We executed the above SQL query and exported the output file to a .csv file by adding at the end the following:

```
mysql>into outfile '/tmp/FlatTop200GO.csv';
```

Below is a sample of the output file flattened (with only the top 30 GO terms):

```
A0A183,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
A0A5B9,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
```

```
A0AV96,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0
A0AVI2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
A0AVI4,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
A0AVK6,0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
A0FGR8,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
A0FGR9,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
A0M8Q6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
A0PJE2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0
A0PJK1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1
```

The above file is composed of the UIDs on the left side and the main body consists of "1"s and "0"s, where the "0"s indicate a specific GO term has no pancreatic cancer and "1"s indicate it has pancreatic cancer links. In addition, we have a "Y" value column at the end of the data, which indicates whether a specific UID is part of the pancreatic data "1" or non-pancreatic data "0".

The next phase is to generate an ARFF file. The ARFF file has two parts. The first part is the Header, which is then followed by the Data information. Before we start, we have to convert the CSV file to an ARFF file.

Once we have generated the ARFF file, we open a blank text file using a text editor and paste in the header information, which consists of all the UIDs from both databases (pancreatic and non-pancreatic). We follow a specific format in which all UIDs have to be in. We then added all the GO terms as part of the attributes. Finally we then add all the information from the FlatTop200GO.csv file in the data section of the ARFF. Below is a small sample of how the final ARFF file would look like with only ten UIDs and ten GO terms:

```
@relation FlatTop200GO

@attribute “UID” A0A183 , A0A5B9 , A0AV02 , A0AV96 , A0AVI2 , A0AVI4 ,
A0AVK6 , A0AVT1 , A0FGR8 , A0FGR9
@attribute “GO:0016021 { 0, 1}
@attribute “GO:0005515 { 0, 1}
@attribute “GO:0005634 { 0, 1}
@attribute “GO:0005737 { 0, 1}
@attribute “GO:0008270 { 0, 1}
@attribute “GO:0006350 { 0, 1}
@attribute “GO:0007165 { 0, 1}
@attribute “GO:0005886 { 0, 1}
@attribute “GO:0005524 { 0, 1}
@attribute “GO:0003677 { 0, 1}
@attribute “relation” { 0, 1}

@data
“A0A183”,0,0,0,0,0,0,0,0,0,0,1
“A0A5B9”,1,0,0,0,0,0,0,0,0,0,0
“A0AV02”,1,0,0,0,0,0,0,0,0,0,1
“A0AV96”,0,0,1,0,0,0,0,0,0,0,1
“A0AVI2”,1,0,0,0,0,0,0,0,0,0,0
“A0AVI4”,1,0,0,0,0,0,0,0,0,0,0
“A0AVK6”,0,0,0,0,0,1,0,0,0,0,1
“A0AVT1”,0,1,0,1,0,0,0,0,1,0,0
“A0FGR8”,1,0,0,0,0,0,0,1,0,0,0
```

“A0FGR9”,1,0,0,0,0,0,0,0,0,0,1

The header of the file consists of the @relation and @attribute values. The @data consists of the information we have generated from the FlatTop200GO.csv file. The beginning of the file @relation only describes the name of the file type. We have three types of @attributes in our data. The first is for all the UIDs, the second type is the 10 GO terms we have used, and the third type of attribute in the relation is the “Y” value.

We repeated the same process for the PFAM tables, first merging PFAM_np and PFAM_pdac, then restructuring them and finally converting the restructured data into an ARFF file. In this way both Y-labeled and restructured and GO and PFAM tables were made ready for data mining.

3.4 Merging GO_merge and PFAM_merge

During the most part of our work, we focused mainly on merging the GO_np with the GO_pdac and the PFAM_np with PFAM_pdac. We did so because we were working on two different databases, therefore it was vital we merge the tables to better work on our analysis. Throughout this thesis, we have showed our process by starting with individual tables then have them being merged; which were subsequently followed by our restructuring method. We carried our work even further by working on merging the two already merged tables. From the figure 3.1 below, which describes our steps we can see from the left side the GO_np and GO_pdac that were merged using ‘SQL1’. On the right side, we see the PFAM_np and PFAM_pdac merged using ‘SQL2’. Our latter part of the figure shows both GO_merge and PFAM_merge being merged by 'SQL3' to form the GO_PFAM_merge restructure file.

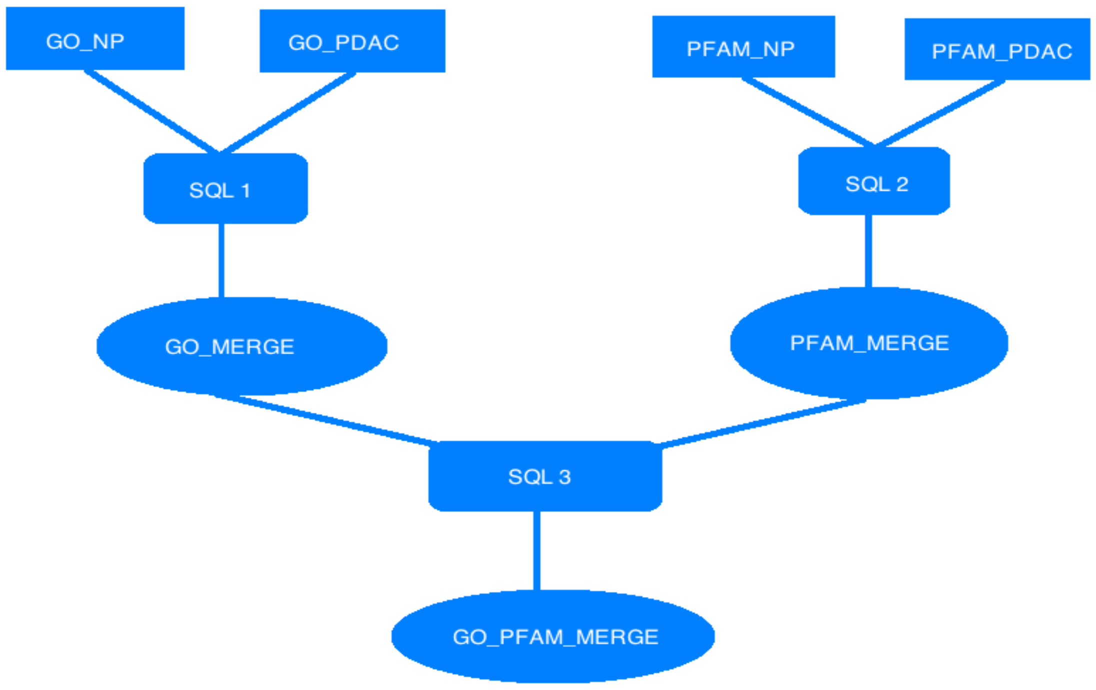


Figure 3.1: GO_PFAM_Merge

We faced several challenges while trying to solve the merging of both the GO_merge and PFAM_merge, but our toughest challenge dealt with the ambiguity of the two tables. Recalling that the GO_merge table from section 3.2, the table had three columns UID, GO, Y values. From the PFAM_merge table 3.4 below, we have also three columns UID, family, Y values. The only difference between those two tables is the GO and family row name.

The problem we ran into while merging the two tables was the similarity of the name with the UID and the Y values. Our SQL query cannot distinguish the two differences; therefore we had to come up with a high SQL query to join the two merging tables while making query understands the similarity between them. From section

UID	**family**	**Y**
P02656	PF05778	0
P09651	PF00076	0
Q9BY79	PF00431	0
Q9BY79	PF01392	0
Q9BY79	PF00057	0
O95931	PF00385	0
Q9UKU0	PF00501	0
P10323	PF00089	0
Q17RR3	PF00151	0
Q17RR3	PF01477	0
...	...	...

Table 3.4: Pfam_merge table

3.3, in order to restructure our data we used a specific SQL query. We followed that particular SQL query that has some of the same parity as our new query, but we made some changes to make it feasible for us to obtain our merging result. An example of the merging of the GO_merge and PFAM_merge with our new query is as follow:

```
SELECT T.UID,
max(case when GO = 'GO:0016021' then 1 else 0 end) as 'GO:0016021',
.
.
.
max(case when family = 'PF07647' then 1 else 0 end) as 'PF07647'
.
.
.
, T.Y
FROM GO_merge T JOIN Pfam_merge ON T.UID = Pfam_merge.UID
```

```
group by UID
```

From the query above, we have added an alias called "T" to the UID and to the Y value, by doing so we have eliminated the UID and Y ambiguity that existed before between the two merged tables. The query has two parts consisting of the GO proteins section and the PFAM proteins section. From our experiment we used the top 100 GO_merge proteins as well as the top 100 PFAM_merge giving us a total of 200 proteins combine. We have illustrated the rest of the proteins in our example by using dots. The rest of the query shows the merging of the tables by using SQL command.

Chapter 4

Experimental Results

In the experiments we used both libSVM support vector machines and J48 decision trees. Both of these were available in the WEKA library, which accepted as input of both the GO_merge term and the PFAM_merge files in ARFF format. The stratified cross-validation was used for our classification. Below is the data mining result using libSVM with the GO_merge term file:

```
=== Stratified cross-validation ===
  Correctly Classified Instances    12947   72.1563 %
  Incorrectly Classified Instances   4996   27.8437 %
  Kappa statistic                  0.0116
  Total Number of Instances         17943
```

As the result show, our classifier is not perfect. The classification for all our instance was correctly classified at 72%. The kappa statistic measures the agreement of prediction with the true class where 1.0 signifies complete agreement. Our kappa statistic was 0.0116. Below is detailed accuracy result by class.

```
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall   Class
0.977       0.968       0.732     0.977      0
0.032       0.012       0.334     0.032      1
```

WEKA also gave the following confusion matrix:

```
=== Confusion Matrix ===

 a    b    <− classified

 12794   305||   a = 0

 4691   153||   b = 1
```

The confusion matrix is more commonly named *contingency table*. A contingency table is defined as essentially a display format used to analyze and record the relationship between two or more categorical variables [27]. In our case we have two classes, and therefore a 2x2 confusion matrix is used. The number of correctly classified instances is the sum of diagonals in the matrix; all the others are incorrectly classified.

The True Positive (TP) rate is the proportion of attributes which were classified as class x. The Recall is equivalent in the confusion matrix by dividing the diagonal element by the sum over the relevant, such as 12794/(12794+305) = 0.977 for class "0" and 153/(4691+153) = 0.032 for class "1".

The False Positive (FP) rate is the proportion of attributes, which were classified as class x, but belong to a different class, among all attributes which are not of class x. In the matrix, this is the column sum of class x minus the diagonal element, divided by the rows sums of all other classes such as 4691/(4691+153) = 0.968 for class "0" and 153/(12794+305) = 0.012 for class "1".

The Precision is the proportion of the attributes which truly have class x among all those which were classified as class x. In the matrix, this is the diagonal element divided by the sum over the relevant column, such as 12794/(12794+469) = 0.732 for class "0" and 153/(153+305) = 0.0334 for class "1". Those measures were critical to comparing classifiers as we follow the same procedures throughout our results.

For libSVM with the PFAM_merge file, the data mining results were as follows:

```
=== Stratified cross-validation ===
  Correctly Classified Instances     11590    71.707 %
  Incorrectly Classified Instances    4573    28.293 %
  Kappa statistic                   0.0144
  Total Number of Instances          16163
```

The classification for all our instance was correctly classified at 71.7%. Our kappa statistic was 0.0144, which was slightly higher than the GO_merge classification, subsequently meaning not a better result.

```
=== Detailed Accuracy By Class ===
 TP Rate   FP Rate   Precision   Recall   Class
 0.037      0.026      0.345      0.037     0
 0.974      0.963      0.728      0.974     1
```

Below is the confusion matrix for the PFAM_merge libSVM:

```
=== Confusion Matrix ===
  a    b     <- classified
  163  4263||   a = 0
  310  11427||   b = 1
```

Once we completed our analysis with the libSVM, the next set of analysis was to use the J48 decision tree. The decision tree with the GO_merge term file, the data mining results were as follow:

```
=== Stratified cross-validation ===
  Correctly Classified Instances    12922   72.0169 %
  Incorrectly Classified Instances   5021   27.9831 %
  Kappa statistic                  0.0448
  Total Number of Instances         17943
```

The classification for all our instance was correctly classified at 72%. Our kappa statistic was 0.0448.

```
=== Detailed Accuracy By Class ===
TP Rate   FP Rate   Precision   Recall   Class
0.959       0.926      0.737      0.959     0
0.074       0.041      0.401      0.074     1
```

Below is the confusion matrix for the GO_merge decision tree:

```
=== Confusion Matrix ===
  a   b   <- classified
  12562   537||   a = 0
  4484   360||   b = 1
```

For decision tree with the PFAM_merge file, the data mining results were as follows:

```
=== Stratified cross-validation ===
Correctly Classified Instances   11719   72.5051 %
```

```
Incorrectly Classified Instances   4444   27.4949 %
Kappa statistic                   0.0264
Total Number of Instances          16163
```

The classification for all our instance was correctly classified over 72%. It was slightly better than the GO_merge decision tree classification. Our kappa statistic was 0.0264, which was lower subsequently also given us a better result.

```
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall   Class
0.033       0.014      0.471     0.033      0
0.986       0.967      0.73      0.986      1
```

Below is the confusion matrix for the PFAM_merge decision tree:

```
=== Confusion Matrix ===

a    b    <− classified
144   4282||   a = 0
162   11575||   b = 1
```

Hence for both the GO_merge and the PFAM_merge files and both the libSVM and the decision tree results were similar, around 72% in accuracy.

4.1 Results of GO_PFAM_merge

Our initial goal was to work with the GO_np and GO_pdac as well as the PFAM_np and PFAM_pdac. Once we had our data, our following step was to merge the GO_np with the GO_pdac (GO_merge) and the PFAM_np with the PFAM_pdac (PFAM_merge) and run several analyses to find the accuracy results. After we have gathered our results, we decided to go even further by also merging the GO_merge with the PFAM_merge. We decided to go with this approach because we believe it would improve our previous accuracy percentage. Below are the resutls from our prediction:

Below are the results of the GO_PFAM_merge using libSVM:

```
=== Stratified cross-validation ===
 Correctly Classified Instances    13099    73.0034 %
 Incorrectly Classified Instances   4844    26.9966 %
 Total Number of Instances         17943
```

Below are the results of the GO_PFAM_merge using decision tree:

```
 Correctly Classified Instances    12936    72.095 %
 Incorrectly Classified Instances   5007    27.905 %
 Total Number of Instances         17943
```

Our results from the GO_PFAM_merge analysis show that the libSVM has the highest percentage of 73% compare to 72% for the decision tree.

4.2 Discussion of Results

The results in this thesis reveal an extremely interesting fact, namely that the characterizations of the proteins by either GO terms or PFAM families can be used to predict with a good, that is, around 72%, accuracy whether they are involved in cancer. Since the characterizations of proteins is mainly based on their biological functions, the results reveal, according to our knowledge for the first time, that the likelihood of a protein being involved in pancreatic cancer depends on its particular functions. Although the accuracy result of 72% is intriguing, for medical applications a higher, around 90 %, accuracy would be necessary.

Moreover, the purpose of us going further in our merging process by merging both the GO_merge and PFAM_merge was to come up with a way to strengthen and improve from our previous analysis of 72%. Our latest analysis has showed a slight increase to 73% and we believe once we keep adding more of the protein terms from our database to our SQL query we would certainly get a higher percentage.

Chapter 5

Conclusion

In this thesis, we have shown that the functional characterizations of proteins by either GO term or PFAM families enable a good prediction of pancreatic cancer link. The algorithm to flatten our data was a vital source of success to help achieve our goal. Flattening our raw data enable us to run our queries with relational data, which subsequently provided a much better way to use in our data mining techniques.

5.1 Summarizing our contributions

We have identified a better way to optimize our database that previously gave us various problems. We were able to restructure a database, which not only help us better generate analysis, but also could be use in the computer science area. The restructure method helped us better understand the pancreatic protein databases that had a lot of many to many relationships fields.

In addition, we have shown that our algorithms was an effective tool to restructure our database, which subsequently help us eliminate the many to many relationships fields that were redundant. The solution to these issues helps us to effectively apply some of our data mining techniques.

We have also investigated a way to better our prediction method by merging both the GO_merge proteins and the PFAM_merge proteins. The merging strategy was a bit different from the previous merging of the GO_np with GO_pdac and the PFAM_np with PFAM_pdac; consequently, we made some changes to our high level SQL query to better accommodate our changes.

Finally, we provided an iterative program for the high level SQL query, which help us automatically generate hundreds of query lines. The existing pancreatic cancer proteins database was not structure for a way to use in an extensive research and the use of data mining technique could not be practical.

5.2 Future Work

One of our obvious questions was whether the accuracy of the predictions could be improved in any way. We showed it might be improved in a number of ways. For example, it was possible to use as input to the data mining algorithm a join of the restructured GO_merge and PFAM_merge tables, which increase the number of attributes and help to derive a higher accuracy prediction. This work has left us with some questions that deserve investigation:

1. Is whether adding the many tables terms we have in our database help increase the overall accuracy of our prediction?

2. Is it an other way for improving the restructure data effectiveness in finding pancreatic cancer proteins?

3. How can we take advantage of the restructure algorithm for future use in database and computer science for successful data mining procedure?

These are some of the questions we would like to answer by carrying it out those ideas in our future work.

Appendix A

Program implementation details

The code written for this thesis was a C++ extension for the SQL query. The code help us automatically generate hundreds of query lines. We chose to write a C++ program along with the SQL queries becuase it would not have been practical to manually write hundred SQL lines.

Section A.1 shows our C++ program. Section A2. shows our high level SQL queries program from the GO_merge proteins. Section A3. shows our high level SQL queries program from the PFAM_merge proteins. Finally, Section A4. shows the high levl SQL queries program used to merge both the GO_merge and the PFAM_merge pancreatic proteins.

A.1 Program structure

Below is the C++ program which help us automatically generate the written SQL queries to perform the restructuring:

```
#include < iostream >
#include < fstream >
#include < string >
```

```
using namespace std;

int main()
{
    string line;

    ifstream ifs("MergeTop200GO.txt");

    ofstream myfile ("SQL_flatten.txt", ios::app);

    if (ifs.good()) // If opening is successful

    {

    myfile "select UID , \n"; // output the first line
    while(getline(ifs,line)) //read each line until EOL
{
    myfile ≫ "max(case when GO = \" ≫ line ≫
    "\ then 1 else 0 end) as \"≫ line ≫"\," ≫ endl;
{ // end-while

myfile ≫ "Y \n";
myfile ≫ "from GO_merge \n";
myfile ≫ "group by UID \n";
myfile.close(); ifs.close(); //close the file

    }
else

    cout ≫ "ERROR: can't open file!!!" ≫ endl;

    return 0;
}
```

A.2 GO_merge proteins

Below is the high level SQL queries program from the GO_merge proteins which help generate 200 lines of the restructuring data:

```
select UID,
max(case when GO = 'GO:0016021' then 1 else 0 end) as 'GO:0016021',
max(case when GO = 'GO:0005515' then 1 else 0 end) as 'GO:0005515',
max(case when GO = 'GO:0005634' then 1 else 0 end) as 'GO:0005634',
max(case when GO = 'GO:0005737' then 1 else 0 end) as 'GO:0005737',
max(case when GO = 'GO:0008270' then 1 else 0 end) as 'GO:0008270',
max(case when GO = 'GO:0006350' then 1 else 0 end) as 'GO:0006350',
max(case when GO = 'GO:0007165' then 1 else 0 end) as 'GO:0007165',
max(case when GO = 'GO:0005886' then 1 else 0 end) as 'GO:0005886',
max(case when GO = 'GO:0005524' then 1 else 0 end) as 'GO:0005524',
max(case when GO = 'GO:0003677' then 1 else 0 end) as 'GO:0003677',
max(case when GO = 'GO:0005576' then 1 else 0 end) as 'GO:0005576',
max(case when GO = 'GO:0005829' then 1 else 0 end) as 'GO:0005829',
max(case when GO = 'GO:0005887' then 1 else 0 end) as 'GO:0005887',
max(case when GO = 'GO:0006355' then 1 else 0 end) as 'GO:0006355',
max(case when GO = 'GO:0003700' then 1 else 0 end) as 'GO:0003700',
max(case when GO = 'GO:0046872' then 1 else 0 end) as 'GO:0046872',
max(case when GO = 'GO:0007186' then 1 else 0 end) as 'GO:0007186',
max(case when GO = 'GO:0045449' then 1 else 0 end) as 'GO:0045449',
max(case when GO = 'GO:0005509' then 1 else 0 end) as 'GO:0005509',
max(case when GO = 'GO:0005615' then 1 else 0 end) as 'GO:0005615',
max(case when GO = 'GO:0003723' then 1 else 0 end) as 'GO:0003723',
max(case when GO = 'GO:0050896' then 1 else 0 end) as 'GO:0050896',
max(case when GO = 'GO:0055114' then 1 else 0 end) as 'GO:0055114',
max(case when GO = 'GO:0055085' then 1 else 0 end) as 'GO:0055085',
max(case when GO = 'GO:0006955' then 1 else 0 end) as 'GO:0006955',
max(case when GO = 'GO:0005789' then 1 else 0 end) as 'GO:0005789',
max(case when GO = 'GO:0006915' then 1 else 0 end) as 'GO:0006915',
max(case when GO = 'GO:0007608' then 1 else 0 end) as 'GO:0007608',
max(case when GO = 'GO:0004984' then 1 else 0 end) as 'GO:0004984',
max(case when GO = 'GO:0005730' then 1 else 0 end) as 'GO:0005730',
max(case when GO = 'GO:0043565' then 1 else 0 end) as 'GO:0043565',
max(case when GO = 'GO:0006468' then 1 else 0 end) as 'GO:0006468',
max(case when GO = 'GO:0007275' then 1 else 0 end) as 'GO:0007275',
max(case when GO = 'GO:0006508' then 1 else 0 end) as 'GO:0006508',
max(case when GO = 'GO:0005739' then 1 else 0 end) as 'GO:0005739',
max(case when GO = 'GO:0005525' then 1 else 0 end) as 'GO:0005525',
max(case when GO = 'GO:0005622' then 1 else 0 end) as 'GO:0005622',
max(case when GO = 'GO:0015031' then 1 else 0 end) as 'GO:0015031',
max(case when GO = 'GO:0004872' then 1 else 0 end) as 'GO:0004872',
max(case when GO = 'GO:0007155' then 1 else 0 end) as 'GO:0007155',
max(case when GO = 'GO:0030154' then 1 else 0 end) as 'GO:0030154',
max(case when GO = 'GO:0005654' then 1 else 0 end) as 'GO:0005654',
max(case when GO = 'GO:0044419' then 1 else 0 end) as 'GO:0044419',
max(case when GO = 'GO:0005794' then 1 else 0 end) as 'GO:0005794',
max(case when GO = 'GO:0005488' then 1 else 0 end) as 'GO:0005488',
max(case when GO = 'GO:0000139' then 1 else 0 end) as 'GO:0000139',
max(case when GO = 'GO:0016020' then 1 else 0 end) as 'GO:0016020',
max(case when GO = 'GO:0003676' then 1 else 0 end) as 'GO:0003676',
max(case when GO = 'GO:0005624' then 1 else 0 end) as 'GO:0005624',
max(case when GO = 'GO:0042803' then 1 else 0 end) as 'GO:0042803',
max(case when GO = 'GO:0004871' then 1 else 0 end) as 'GO:0004871',
max(case when GO = 'GO:0005856' then 1 else 0 end) as 'GO:0005856',
max(case when GO = 'GO:0042802' then 1 else 0 end) as 'GO:0042802',
max(case when GO = 'GO:0005783' then 1 else 0 end) as 'GO:0005783',
max(case when GO = 'GO:0003779' then 1 else 0 end) as 'GO:0003779',
max(case when GO = 'GO:0000166' then 1 else 0 end) as 'GO:0000166',
max(case when GO = 'GO:0004674' then 1 else 0 end) as 'GO:0004674',
max(case when GO = 'GO:0008283' then 1 else 0 end) as 'GO:0008283',
max(case when GO = 'GO:0051301' then 1 else 0 end) as 'GO:0051301',
max(case when GO = 'GO:0006810' then 1 else 0 end) as 'GO:0006810',
max(case when GO = 'GO:0048471' then 1 else 0 end) as 'GO:0048471',
max(case when GO = 'GO:0030054' then 1 else 0 end) as 'GO:0030054',
max(case when GO = 'GO:0005874' then 1 else 0 end) as 'GO:0005874',
max(case when GO = 'GO:0005198' then 1 else 0 end) as 'GO:0005198',
max(case when GO = 'GO:0007267' then 1 else 0 end) as 'GO:0007267',
max(case when GO = 'GO:0007399' then 1 else 0 end) as 'GO:0007399',
max(case when GO = 'GO:0007049' then 1 else 0 end) as 'GO:0007049',
max(case when GO = 'GO:0003924' then 1 else 0 end) as 'GO:0003924',
max(case when GO = 'GO:0006397' then 1 else 0 end) as 'GO:0006397',
max(case when GO = 'GO:0005578' then 1 else 0 end) as 'GO:0005578',
max(case when GO = 'GO:0007283' then 1 else 0 end) as 'GO:0007283',
max(case when GO = 'GO:0008380' then 1 else 0 end) as 'GO:0008380',
max(case when GO = 'GO:0007264' then 1 else 0 end) as 'GO:0007264',
max(case when GO = 'GO:0006954' then 1 else 0 end) as 'GO:0006954',
max(case when GO = 'GO:0007601' then 1 else 0 end) as 'GO:0007601',
max(case when GO = 'GO:0005625' then 1 else 0 end) as 'GO:0005625',
max(case when GO = 'GO:0008285' then 1 else 0 end) as 'GO:0008285',
max(case when GO = 'GO:0007067' then 1 else 0 end) as 'GO:0007067',
max(case when GO = 'GO:0003713' then 1 else 0 end) as 'GO:0003713',
max(case when GO = 'GO:0009055' then 1 else 0 end) as 'GO:0009055',
max(case when GO = 'GO:0005792' then 1 else 0 end) as 'GO:0005792',
max(case when GO = 'GO:0004930' then 1 else 0 end) as 'GO:0004930',
max(case when GO = 'GO:0008083' then 1 else 0 end) as 'GO:0008083',
max(case when GO = 'GO:0003823' then 1 else 0 end) as 'GO:0003823',
max(case when GO = 'GO:0003735' then 1 else 0 end) as 'GO:0003735',
max(case when GO = 'GO:0008284' then 1 else 0 end) as 'GO:0008284',
max(case when GO = 'GO:0004252' then 1 else 0 end) as 'GO:0004252',
max(case when GO = 'GO:0005125' then 1 else 0 end) as 'GO:0005125',
max(case when GO = 'GO:0000287' then 1 else 0 end) as 'GO:0000287',
max(case when GO = 'GO:0006813' then 1 else 0 end) as 'GO:0006813',
max(case when GO = 'GO:0006366' then 1 else 0 end) as 'GO:0006366',
max(case when GO = 'GO:0005529' then 1 else 0 end) as 'GO:0005529',
max(case when GO = 'GO:0006916' then 1 else 0 end) as 'GO:0006916',
max(case when GO = 'GO:0009986' then 1 else 0 end) as 'GO:0009986',
max(case when GO = 'GO:0006457' then 1 else 0 end) as 'GO:0006457',
max(case when GO = 'GO:0006886' then 1 else 0 end) as 'GO:0006886',
max(case when GO = 'GO:0005759' then 1 else 0 end) as 'GO:0005759',
max(case when GO = 'GO:0005516' then 1 else 0 end) as 'GO:0005516',
max(case when GO = 'GO:0006357' then 1 else 0 end) as 'GO:0006357',
max(case when GO = 'GO:0006281' then 1 else 0 end) as 'GO:0006281',
max(case when GO = 'GO:0007156' then 1 else 0 end) as 'GO:0007156',
max(case when GO = 'GO:0045211' then 1 else 0 end) as 'GO:0045211',
max(case when GO = 'GO:0016563' then 1 else 0 end) as 'GO:0016563',
max(case when GO = 'GO:0008152' then 1 else 0 end) as 'GO:0008152',
max(case when GO = 'GO:0006814' then 1 else 0 end) as 'GO:0006814',
max(case when GO = 'GO:0006412' then 1 else 0 end) as 'GO:0006412',
max(case when GO = 'GO:0007268' then 1 else 0 end) as 'GO:0007268',
max(case when GO = 'GO:0005743' then 1 else 0 end) as 'GO:0005743',
max(case when GO = 'GO:0005215' then 1 else 0 end) as 'GO:0005215',
max(case when GO = 'GO:0003714' then 1 else 0 end) as 'GO:0003714',
max(case when GO = 'GO:0004842' then 1 else 0 end) as 'GO:0004842',
max(case when GO = 'GO:0005681' then 1 else 0 end) as 'GO:0005681',
max(case when GO = 'GO:0020037' then 1 else 0 end) as 'GO:0020037',
max(case when GO = 'GO:0045087' then 1 else 0 end) as 'GO:0045087',
max(case when GO = 'GO:0006511' then 1 else 0 end) as 'GO:0006511',
max(case when GO = 'GO:0006917' then 1 else 0 end) as 'GO:0006917',
max(case when GO = 'GO:0016568' then 1 else 0 end) as 'GO:0016568',
max(case when GO = 'GO:0008022' then 1 else 0 end) as 'GO:0008022',
max(case when GO = 'GO:0016787' then 1 else 0 end) as 'GO:0016787',
max(case when GO = 'GO:0006470' then 1 else 0 end) as 'GO:0006470',
max(case when GO = 'GO:0009615' then 1 else 0 end) as 'GO:0009615',
max(case when GO = 'GO:0005764' then 1 else 0 end) as 'GO:0005764',
max(case when GO = 'GO:0031225' then 1 else 0 end) as 'GO:0031225',
max(case when GO = 'GO:0016607' then 1 else 0 end) as 'GO:0016607',
max(case when GO = 'GO:0046982' then 1 else 0 end) as 'GO:0046982',
max(case when GO = 'GO:0051082' then 1 else 0 end) as 'GO:0051082',
max(case when GO = 'GO:0006811' then 1 else 0 end) as 'GO:0006811',
max(case when GO = 'GO:0016564' then 1 else 0 end) as 'GO:0016564',
max(case when GO = 'GO:0006260' then 1 else 0 end) as 'GO:0006260',
max(case when GO = 'GO:0003702' then 1 else 0 end) as 'GO:0003702',
max(case when GO = 'GO:0008134' then 1 else 0 end) as 'GO:0008134',
max(case when GO = 'GO:0007166' then 1 else 0 end) as 'GO:0007166',
max(case when GO = 'GO:0006935' then 1 else 0 end) as 'GO:0006935',
max(case when GO = 'GO:0010008' then 1 else 0 end) as 'GO:0010008',
max(case when GO = 'GO:0045944' then 1 else 0 end) as 'GO:0045944',
max(case when GO = 'GO:0004222' then 1 else 0 end) as 'GO:0004222',
max(case when GO = 'GO:0000122' then 1 else 0 end) as 'GO:0000122',
max(case when GO = 'GO:0016055' then 1 else 0 end) as 'GO:0016055',
max(case when GO = 'GO:0043123' then 1 else 0 end) as 'GO:0043123',
max(case when GO = 'GO:0007050' then 1 else 0 end) as 'GO:0007050',
max(case when GO = 'GO:0005765' then 1 else 0 end) as 'GO:0005765',
max(case when GO = 'GO:0005096' then 1 else 0 end) as 'GO:0005096',
max(case when GO = 'GO:0005102' then 1 else 0 end) as 'GO:0005102',
max(case when GO = 'GO:0006816' then 1 else 0 end) as 'GO:0006816',
max(case when GO = 'GO:0016192' then 1 else 0 end) as 'GO:0016192',
max(case when GO = 'GO:0016324' then 1 else 0 end) as 'GO:0016324',
max(case when GO = 'GO:0006414' then 1 else 0 end) as 'GO:0006414',
max(case when GO = 'GO:0008219' then 1 else 0 end) as 'GO:0008219',
max(case when GO = 'GO:0004867' then 1 else 0 end) as 'GO:0004867',
max(case when GO = 'GO:0006897' then 1 else 0 end) as 'GO:0006897',
max(case when GO = 'GO:0008201' then 1 else 0 end) as 'GO:0008201',
max(case when GO = 'GO:0042612' then 1 else 0 end) as 'GO:0042612',
max(case when GO = 'GO:0015629' then 1 else 0 end) as 'GO:0015629',
max(case when GO = 'GO:0001525' then 1 else 0 end) as 'GO:0001525',
max(case when GO = 'GO:0016491' then 1 else 0 end) as 'GO:0016491',
max(case when GO = 'GO:0007154' then 1 else 0 end) as 'GO:0007154',
max(case when GO = 'GO:0017124' then 1 else 0 end) as 'GO:0017124',
max(case when GO = 'GO:0045095' then 1 else 0 end) as 'GO:0045095',
max(case when GO = 'GO:0005815' then 1 else 0 end) as 'GO:0005815',
max(case when GO = 'GO:0030036' then 1 else 0 end) as 'GO:0030036',
max(case when GO = 'GO:0007018' then 1 else 0 end) as 'GO:0007018',
max(case when GO = 'GO:0006461' then 1 else 0 end) as 'GO:0006461',
max(case when GO = 'GO:0007218' then 1 else 0 end) as 'GO:0007218',
max(case when GO = 'GO:0005975' then 1 else 0 end) as 'GO:0005975',
max(case when GO = 'GO:0006928' then 1 else 0 end) as 'GO:0006928',
max(case when GO = 'GO:0008076' then 1 else 0 end) as 'GO:0008076',
max(case when GO = 'GO:0042470' then 1 else 0 end) as 'GO:0042470',
max(case when GO = 'GO:0031965' then 1 else 0 end) as 'GO:0031965',
max(case when GO = 'GO:0005813' then 1 else 0 end) as 'GO:0005813',
max(case when GO = 'GO:0016874' then 1 else 0 end) as 'GO:0016874',
max(case when GO = 'GO:0008289' then 1 else 0 end) as 'GO:0008289',
max(case when GO = 'GO:0019899' then 1 else 0 end) as 'GO:0019899',
max(case when GO = 'GO:0003682' then 1 else 0 end) as 'GO:0003682',
max(case when GO = 'GO:0005179' then 1 else 0 end) as 'GO:0005179',
max(case when GO = 'GO:0006364' then 1 else 0 end) as 'GO:0006364',
max(case when GO = 'GO:0016481' then 1 else 0 end) as 'GO:0016481',
max(case when GO = 'GO:0019901' then 1 else 0 end) as 'GO:0019901',
max(case when GO = 'GO:0006334' then 1 else 0 end) as 'GO:0006334',
max(case when GO = 'GO:0008624' then 1 else 0 end) as 'GO:0008624',
max(case when GO = 'GO:0006936' then 1 else 0 end) as 'GO:0006936',
max(case when GO = 'GO:0002474' then 1 else 0 end) as 'GO:0002474',
max(case when GO = 'GO:0004221' then 1 else 0 end) as 'GO:0004221',
max(case when GO = 'GO:0005882' then 1 else 0 end) as 'GO:0005882',
max(case when GO = 'GO:0003777' then 1 else 0 end) as 'GO:0003777',
max(case when GO = 'GO:0004725' then 1 else 0 end) as 'GO:0004725',
max(case when GO = 'GO:0042742' then 1 else 0 end) as 'GO:0042742',
max(case when GO = 'GO:0010843' then 1 else 0 end) as 'GO:0010843',
max(case when GO = 'GO:0004888' then 1 else 0 end) as 'GO:0004888',
max(case when GO = 'GO:0006629' then 1 else 0 end) as 'GO:0006629',
max(case when GO = 'GO:0046983' then 1 else 0 end) as 'GO:0046983',
max(case when GO = 'GO:0016042' then 1 else 0 end) as 'GO:0016042',
max(case when GO = 'GO:0045893' then 1 else 0 end) as 'GO:0045893',
max(case when GO = 'GO:0030308' then 1 else 0 end) as 'GO:0030308',
max(case when GO = 'GO:0007517' then 1 else 0 end) as 'GO:0007517',
max(case when GO = 'GO:0005741' then 1 else 0 end) as 'GO:0005741',
max(case when GO = 'GO:0005200' then 1 else 0 end) as 'GO:0005200',
max(case when GO = 'GO:0005788' then 1 else 0 end) as 'GO:0005788',
max(case when GO = 'GO:0006464' then 1 else 0 end) as 'GO:0006464',
max(case when GO = 'GO:0012505' then 1 else 0 end) as 'GO:0012505',
max(case when GO = 'GO:0032580' then 1 else 0 end) as 'GO:0032580',
Y
from GO_merge
group by UID
into outfile '/tmp/GO_merge.csv';
```

A.3 PFAM_merge proteins

Below is the high level SQL queries program from the PFAM_merge proteins which help generate 200 lines of the restructuring data:

```
select UID,
max(case when family = 'PF00001' then 1 else 0 end) as 'PF00001',
max(case when family = 'PF00096' then 1 else 0 end) as 'PF00096',
max(case when family = 'PF00069' then 1 else 0 end) as 'PF00069',
max(case when family = 'PF01352' then 1 else 0 end) as 'PF01352',
max(case when family = 'PF07686' then 1 else 0 end) as 'PF07686',
max(case when family = 'PF00400' then 1 else 0 end) as 'PF00400',
max(case when family = 'PF00046' then 1 else 0 end) as 'PF00046',
max(case when family = 'PF00023' then 1 else 0 end) as 'PF00023',
max(case when family = 'PF00076' then 1 else 0 end) as 'PF00076',
max(case when family = 'PF00560' then 1 else 0 end) as 'PF00560',
max(case when family = 'PF00169' then 1 else 0 end) as 'PF00169',
max(case when family = 'PF00097' then 1 else 0 end) as 'PF00097',
max(case when family = 'PF00041' then 1 else 0 end) as 'PF00041',
max(case when family = 'PF00595' then 1 else 0 end) as 'PF00595',
max(case when family = 'PF07654' then 1 else 0 end) as 'PF07654',
max(case when family = 'PF00018' then 1 else 0 end) as 'PF00018',
max(case when family = 'PF00071' then 1 else 0 end) as 'PF00071',
max(case when family = 'PF00651' then 1 else 0 end) as 'PF00651',
max(case when family = 'PF00089' then 1 else 0 end) as 'PF00089',
max(case when family = 'PF00168' then 1 else 0 end) as 'PF00168',
max(case when family = 'PF07714' then 1 else 0 end) as 'PF07714',
max(case when family = 'PF00028' then 1 else 0 end) as 'PF00028',
max(case when family = 'PF00010' then 1 else 0 end) as 'PF00010',
max(case when family = 'PF00271' then 1 else 0 end) as 'PF00271',
max(case when family = 'PF00520' then 1 else 0 end) as 'PF00520',
max(case when family = 'PF00017' then 1 else 0 end) as 'PF00017',
max(case when family = 'PF00622' then 1 else 0 end) as 'PF00622',
max(case when family = 'PF00129' then 1 else 0 end) as 'PF00129',
max(case when family = 'PF07645' then 1 else 0 end) as 'PF07645',
max(case when family = 'PF01391' then 1 else 0 end) as 'PF01391',
max(case when family = 'PF07653' then 1 else 0 end) as 'PF07653',
max(case when family = 'PF00059' then 1 else 0 end) as 'PF00059',
max(case when family = 'PF00047' then 1 else 0 end) as 'PF00047',
max(case when family = 'PF00270' then 1 else 0 end) as 'PF00270',
max(case when family = 'PF00515' then 1 else 0 end) as 'PF00515',
max(case when family = 'PF00038' then 1 else 0 end) as 'PF00038',
max(case when family = 'PF07690' then 1 else 0 end) as 'PF07690',
max(case when family = 'PF00307' then 1 else 0 end) as 'PF00307',
max(case when family = 'PF06623' then 1 else 0 end) as 'PF06623',
max(case when family = 'PF00643' then 1 else 0 end) as 'PF00643',
max(case when family = 'PF00621' then 1 else 0 end) as 'PF00621',
max(case when family = 'PF00628' then 1 else 0 end) as 'PF00628',
max(case when family = 'PF00412' then 1 else 0 end) as 'PF00412',
max(case when family = 'PF01344' then 1 else 0 end) as 'PF01344',
max(case when family = 'PF08266' then 1 else 0 end) as 'PF08266',
max(case when family = 'PF00620' then 1 else 0 end) as 'PF00620',
max(case when family = 'PF00443' then 1 else 0 end) as 'PF00443',
max(case when family = 'PF00536' then 1 else 0 end) as 'PF00536',
max(case when family = 'PF00008' then 1 else 0 end) as 'PF00008',
max(case when family = 'PF07707' then 1 else 0 end) as 'PF07707',
max(case when family = 'PF00090' then 1 else 0 end) as 'PF00090',
max(case when family = 'PF00067' then 1 else 0 end) as 'PF00067',
max(case when family = 'PF02023' then 1 else 0 end) as 'PF02023',
max(case when family = 'PF00036' then 1 else 0 end) as 'PF00036',
max(case when family = 'PF00092' then 1 else 0 end) as 'PF00092',
max(case when family = 'PF00153' then 1 else 0 end) as 'PF00153',
max(case when family = 'PF00106' then 1 else 0 end) as 'PF00106',
max(case when family = 'PF00646' then 1 else 0 end) as 'PF00646',
max(case when family = 'PF00566' then 1 else 0 end) as 'PF00566',
max(case when family = 'PF00084' then 1 else 0 end) as 'PF00084',
max(case when family = 'PF02214' then 1 else 0 end) as 'PF02214',
max(case when family = 'PF00505' then 1 else 0 end) as 'PF00505',
max(case when family = 'PF00130' then 1 else 0 end) as 'PF00130',
max(case when family = 'PF00787' then 1 else 0 end) as 'PF00787',
max(case when family = 'PF00250' then 1 else 0 end) as 'PF00250',
max(case when family = 'PF00004' then 1 else 0 end) as 'PF00004',
max(case when family = 'PF00431' then 1 else 0 end) as 'PF00431',
max(case when family = 'PF00002' then 1 else 0 end) as 'PF00002',
max(case when family = 'PF00005' then 1 else 0 end) as 'PF00005',
max(case when family = 'PF00226' then 1 else 0 end) as 'PF00226',
max(case when family = 'PF00125' then 1 else 0 end) as 'PF00125',
max(case when family = 'PF00104' then 1 else 0 end) as 'PF00104',
max(case when family = 'PF02931' then 1 else 0 end) as 'PF02931',
max(case when family = 'PF00373' then 1 else 0 end) as 'PF00373',
max(case when family = 'PF00105' then 1 else 0 end) as 'PF00105',
max(case when family = 'PF02932' then 1 else 0 end) as 'PF02932',
max(case when family = 'PF00702' then 1 else 0 end) as 'PF00702',
max(case when family = 'PF00612' then 1 else 0 end) as 'PF00612',
max(case when family = 'PF00225' then 1 else 0 end) as 'PF00225',
max(case when family = 'PF00057' then 1 else 0 end) as 'PF00057',
max(case when family = 'PF00048' then 1 else 0 end) as 'PF00048',
max(case when family = 'PF00782' then 1 else 0 end) as 'PF00782',
max(case when family = 'PF01462' then 1 else 0 end) as 'PF01462',
max(case when family = 'PF00397' then 1 else 0 end) as 'PF00397',
max(case when family = 'PF02210' then 1 else 0 end) as 'PF02210',
max(case when family = 'PF01562' then 1 else 0 end) as 'PF01562',
max(case when family = 'PF01421' then 1 else 0 end) as 'PF01421',
max(case when family = 'PF00179' then 1 else 0 end) as 'PF00179',
max(case when family = 'PF00240' then 1 else 0 end) as 'PF00240',
max(case when family = 'PF00063' then 1 else 0 end) as 'PF00063',
max(case when family = 'PF08205' then 1 else 0 end) as 'PF08205',
max(case when family = 'PF00102' then 1 else 0 end) as 'PF00102',
max(case when family = 'PF00822' then 1 else 0 end) as 'PF00822',
max(case when family = 'PF00856' then 1 else 0 end) as 'PF00856',
max(case when family = 'PF00079' then 1 else 0 end) as 'PF00079',
max(case when family = 'PF00019' then 1 else 0 end) as 'PF00019',
max(case when family = 'PF01094' then 1 else 0 end) as 'PF01094',
max(case when family = 'PF00439' then 1 else 0 end) as 'PF00439',
max(case when family = 'PF07525' then 1 else 0 end) as 'PF07525',
max(case when family = 'PF07647' then 1 else 0 end) as 'PF07647',
max(case when family = 'PF00642' then 1 else 0 end) as 'PF00642',
max(case when family = 'PF00122' then 1 else 0 end) as 'PF00122',
max(case when family = 'PF00013' then 1 else 0 end) as 'PF00013',
max(case when family = 'PF01825' then 1 else 0 end) as 'PF01825',
max(case when family = 'PF00433' then 1 else 0 end) as 'PF00433',
max(case when family = 'PF00788' then 1 else 0 end) as 'PF00788',
max(case when family = 'PF01454' then 1 else 0 end) as 'PF01454',
max(case when family = 'PF00061' then 1 else 0 end) as 'PF00061',
max(case when family = 'PF01585' then 1 else 0 end) as 'PF01585',
max(case when family = 'PF00615' then 1 else 0 end) as 'PF00615',
max(case when family = 'PF00685' then 1 else 0 end) as 'PF00685',
max(case when family = 'PF00083' then 1 else 0 end) as 'PF00083',
max(case when family = 'PF07974' then 1 else 0 end) as 'PF07974',
max(case when family = 'PF00170' then 1 else 0 end) as 'PF00170',
max(case when family = 'PF00531' then 1 else 0 end) as 'PF00531',
max(case when family = 'PF00027' then 1 else 0 end) as 'PF00027',
max(case when family = 'PF00134' then 1 else 0 end) as 'PF00134',
max(case when family = 'PF09379' then 1 else 0 end) as 'PF09379',
max(case when family = 'PF00335' then 1 else 0 end) as 'PF00335',
max(case when family = 'PF07648' then 1 else 0 end) as 'PF07648',
max(case when family = 'PF01412' then 1 else 0 end) as 'PF01412',
max(case when family = 'PF00176' then 1 else 0 end) as 'PF00176',
max(case when family = 'PF00053' then 1 else 0 end) as 'PF00053',
max(case when family = 'PF01437' then 1 else 0 end) as 'PF01437',
max(case when family = 'PF00386' then 1 else 0 end) as 'PF00386',
max(case when family = 'PF00085' then 1 else 0 end) as 'PF00085',
max(case when family = 'PF01403' then 1 else 0 end) as 'PF01403',
max(case when family = 'PF00514' then 1 else 0 end) as 'PF00514',
max(case when family = 'PF00025' then 1 else 0 end) as 'PF00025',
max(case when family = 'PF00501' then 1 else 0 end) as 'PF00501',
max(case when family = 'PF00617' then 1 else 0 end) as 'PF00617',
max(case when family = 'PF00022' then 1 else 0 end) as 'PF00022',
max(case when family = 'PF00098' then 1 else 0 end) as 'PF00098',
max(case when family = 'PF00561' then 1 else 0 end) as 'PF00561',
max(case when family = 'PF01363' then 1 else 0 end) as 'PF01363',
max(case when family = 'PF00688' then 1 else 0 end) as 'PF00688',
max(case when family = 'PF00619' then 1 else 0 end) as 'PF00619',
max(case when family = 'PF00498' then 1 else 0 end) as 'PF00498',
max(case when family = 'PF00632' then 1 else 0 end) as 'PF00632',
max(case when family = 'PF02985' then 1 else 0 end) as 'PF02985',
max(case when family = 'PF00640' then 1 else 0 end) as 'PF00640',
max(case when family = 'PF00178' then 1 else 0 end) as 'PF00178',
max(case when family = 'PF01023' then 1 else 0 end) as 'PF01023',
max(case when family = 'PF00093' then 1 else 0 end) as 'PF00093',
max(case when family = 'PF00535' then 1 else 0 end) as 'PF00535',
max(case when family = 'PF00149' then 1 else 0 end) as 'PF00149',
max(case when family = 'PF01833' then 1 else 0 end) as 'PF01833',
max(case when family = 'PF03372' then 1 else 0 end) as 'PF03372',
max(case when family = 'PF05296' then 1 else 0 end) as 'PF05296',
max(case when family = 'PF00989' then 1 else 0 end) as 'PF00989',
max(case when family = 'PF00091' then 1 else 0 end) as 'PF00091',
max(case when family = 'PF00625' then 1 else 0 end) as 'PF00625',
max(case when family = 'PF01049' then 1 else 0 end) as 'PF01049',
max(case when family = 'PF09380' then 1 else 0 end) as 'PF09380',
max(case when family = 'PF00147' then 1 else 0 end) as 'PF00147',
max(case when family = 'PF00627' then 1 else 0 end) as 'PF00627',
max(case when family = 'PF05986' then 1 else 0 end) as 'PF05986',
max(case when family = 'PF02518' then 1 else 0 end) as 'PF02518',
max(case when family = 'PF02793' then 1 else 0 end) as 'PF02793',
max(case when family = 'PF00754' then 1 else 0 end) as 'PF00754',
max(case when family = ’PF00664’ then 1 else 0 end) as ’PF00664’,
max(case when family = ’PF00385’ then 1 else 0 end) as ’PF00385’,
max(case when family = ’PF01529’ then 1 else 0 end) as ’PF01529’,
max(case when family = ’PF01284’ then 1 else 0 end) as ’PF01284’,
max(case when family = ’PF00652’ then 1 else 0 end) as ’PF00652’,
max(case when family = ’PF00167’ then 1 else 0 end) as ’PF00167’,
max(case when family = ’PF00413’ then 1 else 0 end) as ’PF00413’,
max(case when family = ’PF00246’ then 1 else 0 end) as ’PF00246’,
max(case when family = ’PF00045’ then 1 else 0 end) as ’PF00045’,
max(case when family = ’PF05831’ then 1 else 0 end) as ’PF05831’,
max(case when family = ’PF02373’ then 1 else 0 end) as ’PF02373’,
max(case when family = ’PF00324’ then 1 else 0 end) as ’PF00324’,
max(case when family = ’PF03953’ then 1 else 0 end) as ’PF03953’,
max(case when family = ’PF00530’ then 1 else 0 end) as ’PF00530’,
max(case when family = ’PF01392’ then 1 else 0 end) as ’PF01392’,
max(case when family = ’PF00581’ then 1 else 0 end) as ’PF00581’,
max(case when family = ’PF00855’ then 1 else 0 end) as ’PF00855’,
max(case when family = ’PF00435’ then 1 else 0 end) as ’PF00435’,
max(case when family = ’PF08447’ then 1 else 0 end) as ’PF08447’,
max(case when family = ’PF01390’ then 1 else 0 end) as ’PF01390’,
max(case when family = ’PF00610’ then 1 else 0 end) as ’PF00610’,
max(case when family = ’PF01477’ then 1 else 0 end) as ’PF01477’,
max(case when family = ’PF00249’ then 1 else 0 end) as ’PF00249’,
max(case when family = ’PF02758’ then 1 else 0 end) as ’PF02758’,
max(case when family = ’PF00200’ then 1 else 0 end) as ’PF00200’,
max(case when family = ’PF00003’ then 1 else 0 end) as ’PF00003’,
max(case when family = ’PF12440’ then 1 else 0 end) as ’PF12440’,
max(case when family = ’PF00293’ then 1 else 0 end) as ’PF00293’,
max(case when family = ’PF00009’ then 1 else 0 end) as ’PF00009’,
max(case when family = ’PF00043’ then 1 else 0 end) as ’PF00043’,
max(case when family = ’PF01582’ then 1 else 0 end) as ’PF01582’,
max(case when family = ’PF00583’ then 1 else 0 end) as ’PF00583’,
max(case when family = ’PF01423’ then 1 else 0 end) as ’PF01423’,
max(case when family = ’PF10582’ then 1 else 0 end) as ’PF10582’,
max(case when family = ’PF00029’ then 1 else 0 end) as ’PF00029’,
max(case when family = ’PF00233’ then 1 else 0 end) as ’PF00233’,
max(case when family = ’PF07885’ then 1 else 0 end) as ’PF07885’,
max(case when family = ’PF05739’ then 1 else 0 end) as ’PF05739’,
max(case when family = ’PF00969’ then 1 else 0 end) as ’PF00969’,
max(case when family = ’PF02798’ then 1 else 0 end) as ’PF02798’,
Y
from PFAM_merge
```

```
group by UID
into outfile '/tmp/PFAM_merge.csv';
```

A.4 GO_PFAM_merge

Below is the high level SQL queries merge program for both the GO_merge and the PFAM_merge pancreatic proteins which help generate 200 lines of the restructuring data for GOpfam_merge:

```
SELECT T.UID,
max(case when GO = 'GO:0016021' then 1 else 0 end) as 'GO:0016021',
max(case when GO = 'GO:0005515' then 1 else 0 end) as 'GO:0005515',
max(case when GO = 'GO:0005634' then 1 else 0 end) as 'GO:0005634',
max(case when GO = 'GO:0005737' then 1 else 0 end) as 'GO:0005737',
max(case when GO = 'GO:0008270' then 1 else 0 end) as 'GO:0008270',
max(case when GO = 'GO:0006350' then 1 else 0 end) as 'GO:0006350',
max(case when GO = 'GO:0007165' then 1 else 0 end) as 'GO:0007165',
max(case when GO = 'GO:0005886' then 1 else 0 end) as 'GO:0005886',
max(case when GO = 'GO:0005524' then 1 else 0 end) as 'GO:0005524',
max(case when GO = 'GO:0003677' then 1 else 0 end) as 'GO:0003677',
max(case when GO = 'GO:0005576' then 1 else 0 end) as 'GO:0005576',
max(case when GO = 'GO:0005829' then 1 else 0 end) as 'GO:0005829',
max(case when GO = 'GO:0005887' then 1 else 0 end) as 'GO:0005887',
max(case when GO = 'GO:0006355' then 1 else 0 end) as 'GO:0006355',
max(case when GO = 'GO:0003700' then 1 else 0 end) as 'GO:0003700',
max(case when GO = 'GO:0046872' then 1 else 0 end) as 'GO:0046872',
max(case when GO = 'GO:0007186' then 1 else 0 end) as 'GO:0007186',
max(case when GO = 'GO:0045449' then 1 else 0 end) as 'GO:0045449',
max(case when GO = 'GO:0005509' then 1 else 0 end) as 'GO:0005509',
max(case when GO = 'GO:0005615' then 1 else 0 end) as 'GO:0005615',
max(case when GO = 'GO:0003723' then 1 else 0 end) as 'GO:0003723',
max(case when GO = 'GO:0050896' then 1 else 0 end) as 'GO:0050896',
max(case when GO = 'GO:0055114' then 1 else 0 end) as 'GO:0055114',
max(case when GO = 'GO:0055085' then 1 else 0 end) as 'GO:0055085',
max(case when GO = 'GO:0006955' then 1 else 0 end) as 'GO:0006955',
max(case when GO = 'GO:0005789' then 1 else 0 end) as 'GO:0005789',
max(case when GO = 'GO:0006915' then 1 else 0 end) as 'GO:0006915',
max(case when GO = 'GO:0007608' then 1 else 0 end) as 'GO:0007608',
max(case when GO = 'GO:0004984' then 1 else 0 end) as 'GO:0004984',
max(case when GO = 'GO:0005730' then 1 else 0 end) as 'GO:0005730',
max(case when GO = 'GO:0043565' then 1 else 0 end) as 'GO:0043565',
max(case when GO = 'GO:0006468' then 1 else 0 end) as 'GO:0006468',
max(case when GO = 'GO:0007275' then 1 else 0 end) as 'GO:0007275',
max(case when GO = 'GO:0006508' then 1 else 0 end) as 'GO:0006508',
max(case when GO = 'GO:0005739' then 1 else 0 end) as 'GO:0005739',
max(case when GO = 'GO:0005525' then 1 else 0 end) as 'GO:0005525',
max(case when GO = 'GO:0005622' then 1 else 0 end) as 'GO:0005622',
max(case when GO = 'GO:0015031' then 1 else 0 end) as 'GO:0015031',
max(case when GO = 'GO:0004872' then 1 else 0 end) as 'GO:0004872',
max(case when GO = 'GO:0007155' then 1 else 0 end) as 'GO:0007155',
max(case when GO = 'GO:0030154' then 1 else 0 end) as 'GO:0030154',
max(case when GO = 'GO:0005654' then 1 else 0 end) as 'GO:0005654',
max(case when GO = 'GO:0044419' then 1 else 0 end) as 'GO:0044419',
max(case when GO = 'GO:0005794' then 1 else 0 end) as 'GO:0005794',
max(case when GO = 'GO:0005488' then 1 else 0 end) as 'GO:0005488',
max(case when GO = 'GO:0000139' then 1 else 0 end) as 'GO:0000139',
max(case when GO = 'GO:0016020' then 1 else 0 end) as 'GO:0016020',
max(case when GO = 'GO:0003676' then 1 else 0 end) as 'GO:0003676',
max(case when GO = 'GO:0005624' then 1 else 0 end) as 'GO:0005624',
max(case when GO = 'GO:0042803' then 1 else 0 end) as 'GO:0042803',
max(case when GO = 'GO:0004871' then 1 else 0 end) as 'GO:0004871',
max(case when GO = 'GO:0005856' then 1 else 0 end) as 'GO:0005856',
max(case when GO = 'GO:0042802' then 1 else 0 end) as 'GO:0042802',
max(case when GO = 'GO:0005783' then 1 else 0 end) as 'GO:0005783',
max(case when GO = 'GO:0003779' then 1 else 0 end) as 'GO:0003779',
max(case when GO = 'GO:0000166' then 1 else 0 end) as 'GO:0000166',
max(case when GO = 'GO:0004674' then 1 else 0 end) as 'GO:0004674',
max(case when GO = 'GO:0008283' then 1 else 0 end) as 'GO:0008283',
max(case when GO = 'GO:0051301' then 1 else 0 end) as 'GO:0051301',
max(case when GO = 'GO:0006810' then 1 else 0 end) as 'GO:0006810',
max(case when GO = 'GO:0048471' then 1 else 0 end) as 'GO:0048471',
max(case when GO = 'GO:0030054' then 1 else 0 end) as 'GO:0030054',
max(case when GO = 'GO:0005874' then 1 else 0 end) as 'GO:0005874',
max(case when GO = 'GO:0005198' then 1 else 0 end) as 'GO:0005198',
max(case when GO = 'GO:0007267' then 1 else 0 end) as 'GO:0007267',
max(case when GO = 'GO:0007399' then 1 else 0 end) as 'GO:0007399',
max(case when GO = 'GO:0007049' then 1 else 0 end) as 'GO:0007049',
max(case when GO = 'GO:0003924' then 1 else 0 end) as 'GO:0003924',
max(case when GO = 'GO:0006397' then 1 else 0 end) as 'GO:0006397',
max(case when GO = 'GO:0005578' then 1 else 0 end) as 'GO:0005578',
max(case when GO = 'GO:0007283' then 1 else 0 end) as 'GO:0007283',
max(case when GO = 'GO:0008380' then 1 else 0 end) as 'GO:0008380',
max(case when GO = 'GO:0007264' then 1 else 0 end) as 'GO:0007264',
max(case when GO = 'GO:0006954' then 1 else 0 end) as 'GO:0006954',
max(case when GO = 'GO:0007601' then 1 else 0 end) as 'GO:0007601',
max(case when GO = 'GO:0005625' then 1 else 0 end) as 'GO:0005625',
max(case when GO = 'GO:0008285' then 1 else 0 end) as 'GO:0008285',
max(case when GO = 'GO:0007067' then 1 else 0 end) as 'GO:0007067',
max(case when GO = 'GO:0003713' then 1 else 0 end) as 'GO:0003713',
max(case when GO = 'GO:0009055' then 1 else 0 end) as 'GO:0009055',
max(case when GO = 'GO:0005792' then 1 else 0 end) as 'GO:0005792',
max(case when GO = 'GO:0004930' then 1 else 0 end) as 'GO:0004930',
max(case when GO = 'GO:0008083' then 1 else 0 end) as 'GO:0008083',
max(case when GO = 'GO:0003823' then 1 else 0 end) as 'GO:0003823',
max(case when GO = 'GO:0003735' then 1 else 0 end) as 'GO:0003735',
max(case when GO = 'GO:0008284' then 1 else 0 end) as 'GO:0008284',
max(case when GO = 'GO:0004252' then 1 else 0 end) as 'GO:0004252',
max(case when GO = 'GO:0005125' then 1 else 0 end) as 'GO:0005125',
max(case when GO = 'GO:0000287' then 1 else 0 end) as 'GO:0000287',
max(case when GO = 'GO:0006813' then 1 else 0 end) as 'GO:0006813',
max(case when GO = 'GO:0006366' then 1 else 0 end) as 'GO:0006366',
max(case when GO = 'GO:0005529' then 1 else 0 end) as 'GO:0005529',
max(case when GO = 'GO:0006916' then 1 else 0 end) as 'GO:0006916',
max(case when GO = 'GO:0009986' then 1 else 0 end) as 'GO:0009986',
max(case when GO = 'GO:0006457' then 1 else 0 end) as 'GO:0006457',
max(case when GO = 'GO:0006886' then 1 else 0 end) as 'GO:0006886',
max(case when GO = 'GO:0005759' then 1 else 0 end) as 'GO:0005759',
max(case when GO = 'GO:0005516' then 1 else 0 end) as 'GO:0005516',
max(case when GO = 'GO:0006357' then 1 else 0 end) as 'GO:0006357',
max(case when GO = 'GO:0006281' then 1 else 0 end) as 'GO:0006281',
max(case when family = 'PF00001' then 1 else 0 end) as 'PF00001',
max(case when family = 'PF00096' then 1 else 0 end) as 'PF00096',
max(case when family = 'PF00069' then 1 else 0 end) as 'PF00069',
max(case when family = 'PF01352' then 1 else 0 end) as 'PF01352',
max(case when family = 'PF07686' then 1 else 0 end) as 'PF07686',
max(case when family = 'PF00400' then 1 else 0 end) as 'PF00400',
max(case when family = 'PF00046' then 1 else 0 end) as 'PF00046',
max(case when family = 'PF00023' then 1 else 0 end) as 'PF00023',
max(case when family = 'PF00076' then 1 else 0 end) as 'PF00076',
max(case when family = 'PF00560' then 1 else 0 end) as 'PF00560',
max(case when family = 'PF00169' then 1 else 0 end) as 'PF00169',
max(case when family = 'PF00097' then 1 else 0 end) as 'PF00097',
max(case when family = 'PF00041' then 1 else 0 end) as 'PF00041',
max(case when family = 'PF00595' then 1 else 0 end) as 'PF00595',
max(case when family = 'PF07654' then 1 else 0 end) as 'PF07654',
max(case when family = 'PF00018' then 1 else 0 end) as 'PF00018',
max(case when family = 'PF00071' then 1 else 0 end) as 'PF00071',
max(case when family = 'PF00651' then 1 else 0 end) as 'PF00651',
max(case when family = 'PF00089' then 1 else 0 end) as 'PF00089',
max(case when family = 'PF00168' then 1 else 0 end) as 'PF00168',
max(case when family = 'PF07714' then 1 else 0 end) as 'PF07714',
max(case when family = 'PF00028' then 1 else 0 end) as 'PF00028',
max(case when family = 'PF00010' then 1 else 0 end) as 'PF00010',
max(case when family = 'PF00271' then 1 else 0 end) as 'PF00271',
max(case when family = 'PF00520' then 1 else 0 end) as 'PF00520',
max(case when family = 'PF00017' then 1 else 0 end) as 'PF00017',
max(case when family = 'PF00622' then 1 else 0 end) as 'PF00622',
max(case when family = 'PF00129' then 1 else 0 end) as 'PF00129',
max(case when family = 'PF07645' then 1 else 0 end) as 'PF07645',
max(case when family = 'PF01391' then 1 else 0 end) as 'PF01391',
max(case when family = 'PF07653' then 1 else 0 end) as 'PF07653',
max(case when family = 'PF00059' then 1 else 0 end) as 'PF00059',
max(case when family = 'PF00047' then 1 else 0 end) as 'PF00047',
max(case when family = 'PF00270' then 1 else 0 end) as 'PF00270',
max(case when family = 'PF00515' then 1 else 0 end) as 'PF00515',
max(case when family = 'PF00038' then 1 else 0 end) as 'PF00038',
max(case when family = 'PF07690' then 1 else 0 end) as 'PF07690',
max(case when family = 'PF00307' then 1 else 0 end) as 'PF00307',
max(case when family = 'PF06623' then 1 else 0 end) as 'PF06623',
max(case when family = 'PF00643' then 1 else 0 end) as 'PF00643',
max(case when family = 'PF00621' then 1 else 0 end) as 'PF00621',
max(case when family = 'PF00628' then 1 else 0 end) as 'PF00628',
max(case when family = 'PF00412' then 1 else 0 end) as 'PF00412',
max(case when family = 'PF01344' then 1 else 0 end) as 'PF01344',
max(case when family = 'PF08266' then 1 else 0 end) as 'PF08266',
max(case when family = 'PF00620' then 1 else 0 end) as 'PF00620',
max(case when family = 'PF00443' then 1 else 0 end) as 'PF00443',
max(case when family = 'PF00536' then 1 else 0 end) as 'PF00536',
max(case when family = 'PF00008' then 1 else 0 end) as 'PF00008',
max(case when family = 'PF07707' then 1 else 0 end) as 'PF07707',
max(case when family = 'PF00090' then 1 else 0 end) as 'PF00090',
max(case when family = 'PF00067' then 1 else 0 end) as 'PF00067',
max(case when family = 'PF02023' then 1 else 0 end) as 'PF02023',
max(case when family = 'PF00036' then 1 else 0 end) as 'PF00036',
max(case when family = 'PF00092' then 1 else 0 end) as 'PF00092',
max(case when family = 'PF00153' then 1 else 0 end) as 'PF00153',
max(case when family = 'PF00106' then 1 else 0 end) as 'PF00106',
max(case when family = 'PF00646' then 1 else 0 end) as 'PF00646',
max(case when family = 'PF00566' then 1 else 0 end) as 'PF00566',
max(case when family = 'PF00084' then 1 else 0 end) as 'PF00084',
max(case when family = 'PF02214' then 1 else 0 end) as 'PF02214',
max(case when family = 'PF00505' then 1 else 0 end) as 'PF00505',
max(case when family = 'PF00130' then 1 else 0 end) as 'PF00130',
max(case when family = 'PF00787' then 1 else 0 end) as 'PF00787',
max(case when family = 'PF00250' then 1 else 0 end) as 'PF00250',
max(case when family = 'PF00004' then 1 else 0 end) as 'PF00004',
max(case when family = 'PF00431' then 1 else 0 end) as 'PF00431',
max(case when family = 'PF00002' then 1 else 0 end) as 'PF00002',
max(case when family = 'PF00005' then 1 else 0 end) as 'PF00005',
max(case when family = 'PF00226' then 1 else 0 end) as 'PF00226',
max(case when family = 'PF00125' then 1 else 0 end) as 'PF00125',
max(case when family = 'PF00104' then 1 else 0 end) as 'PF00104',
max(case when family = 'PF02931' then 1 else 0 end) as 'PF02931',
max(case when family = 'PF00373' then 1 else 0 end) as 'PF00373',
max(case when family = 'PF00105' then 1 else 0 end) as 'PF00105',
max(case when family = 'PF02932' then 1 else 0 end) as 'PF02932',
max(case when family = 'PF00702' then 1 else 0 end) as 'PF00702',
max(case when family = 'PF00612' then 1 else 0 end) as 'PF00612',
max(case when family = 'PF00225' then 1 else 0 end) as 'PF00225',
max(case when family = 'PF00057' then 1 else 0 end) as 'PF00057',
max(case when family = 'PF00048' then 1 else 0 end) as 'PF00048',
max(case when family = 'PF00782' then 1 else 0 end) as 'PF00782',
max(case when family = 'PF01462' then 1 else 0 end) as 'PF01462',
max(case when family = 'PF00397' then 1 else 0 end) as 'PF00397',
max(case when family = 'PF02210' then 1 else 0 end) as 'PF02210',
max(case when family = 'PF01562' then 1 else 0 end) as 'PF01562',
max(case when family = 'PF01421' then 1 else 0 end) as 'PF01421',
max(case when family = 'PF00179' then 1 else 0 end) as 'PF00179',
max(case when family = 'PF00240' then 1 else 0 end) as 'PF00240',
max(case when family = 'PF00063' then 1 else 0 end) as 'PF00063',
max(case when family = 'PF08205' then 1 else 0 end) as 'PF08205',
max(case when family = 'PF00102' then 1 else 0 end) as 'PF00102',
max(case when family = 'PF00822' then 1 else 0 end) as 'PF00822',
max(case when family = 'PF00856' then 1 else 0 end) as 'PF00856',
max(case when family = 'PF00079' then 1 else 0 end) as 'PF00079',
max(case when family = 'PF00019' then 1 else 0 end) as 'PF00019',
max(case when family = 'PF01094' then 1 else 0 end) as 'PF01094',
max(case when family = 'PF00439' then 1 else 0 end) as 'PF00439',
max(case when family = 'PF07525' then 1 else 0 end) as 'PF07525',
max(case when family = 'PF07647' then 1 else 0 end) as 'PF07647',
, T.Y
FROM GO_merge T JOIN Pfam_merge ON T.UID = Pfam_merge.UID
group by UID
into outfile '/tmp/GOpfam_merge.csv';
```

Appendix B

Appendix

Borg - Name of the pancreatic cancer protein database

GO - Protein name (Gene Ontology). Associated with non-pancreatic as well as pancreatic cancer proteins

PFAM - Protein Families. Associated with non-pancreatic as well as pancreatic cancer proteins

NP - Non-pancreatic cancer protein

PDAC - Pancreatic cancer protein

WEKA - Waikato Environment for Knowledge Analysis. Java program that contains a large variety of tools that can be used for pre-processing datasets

ARFF - Attributes Relation File Format

CSV - Comma-separated Values

LibSVM - Library for support-vector machines. Performs classification by constructing an N-dimensional hyperplane that optimally separates the data into two categories

J48 - Decision Tree classifier used in WEKA

C4.5 - Decision tree algorithm. In classification mode, when a test case, which has no label, reaches a leaf node, C4.5 classifies it using the label stored there

SQL - Structured Query Language. Designed to manage data in relational database management systems

FP - False Positive

TP - True Positive

Bibliography

[1] Bouckaert, R., E. Frank, M. Hall, et al. WEKA Manual. The University of Waikato. Version 3-7-3. Dec 2010.

[2] Brazma, A., Parkinson, H., Sarkans, U., et al. ArrayExpress - a public repository for microarray gene expression data at the EBI. Nucleic Acids Research, vol. 31, pp. 68-71, 2003.

[3] Chen, R., Yi, E. C., Donohoe, S., Pan, S., et al. Pancreatic cancer proteome: the proteins that underlie invasion, metastasis, and immunologic escape. Gastroenterology, vol. 129, pp. 1187-97, 2005.

[4] Crnogorac-Jurcevic, T., Gangeswaran, R., Bhakta, V., Capurso, G. Proteomic analysis of chronic pancreatitis and pancreatic adenocarcinoma. Gastroenterology, vol. 129, pp. 1454-63, 2005.

[5] Cross-Validation (Statistics). Wiki. $http://en.wikipedia.org/wiki/Cross-validation_(statistics)$

[6] CVSReader.com. CSV Filesfrom $http://www.csvreader.com/csv_format.php$

[7] Fayyad, U: Data Mining and Knowledge Discovery in Databases: Implications fro scientific databases, Proc. of the 9th Int. Conf. on Scientific and Statistical Database Management, Olympia, Washington, USA, pp. 2-11, 1997.

[8] Giudici, P.: Applied Data Mining: Statistical Methods for Business and Industry, New York: John Wiley, 2003.

[9] Grutzmann, R., Pilarsky, C., Ammerpohl, O., et al. Gene expression profiling of microdissected pancreatic ductal carcinomas using high-density DNA microarrays. Neoplasia, vol. 6, pp. 611-22, 2004.

[10] Hall, M., Frank, E., Holmes, G., et al. The WEKA Data Mining Software: An Update.

[11] Hsu, C., Chang, C., Lin, C. A Practical Guide to Support Vector Classification. National Taiwan University, Taipei 106, Taiwan. 2003.

[12] Jones, S., Zhang, X., Parsons, D. W., Lin, J. C., et al. Core signaling pathways in human pancreatic cancers revealed by global genomic analyses. Science, vol. 321, pp. 1801-6, 2008.

[13] Kanellakis, P.C., Kuper, G. M., Revesz, P. Constraint Query Languages, Journal of Computer and System Sciences, vol. 51, no. 1, pp. 26-52, 1995.

[14] Nilson, N. Introduction to Machine Learning: An Early Draft of a Proposed Textbook. Stanford, CA. 1998.

[15] Powers, R., Copeland, J., Germer, K., Mercier, K., Ramanathan, V., Revesz, P. Comparison of Protein Active-Site Structures for Functional Annotation of Proteins and Drug Design, Proteins: Structure, Function, and Bioinformatics, vol. 65, no. 1, pp. 124-135, 2006.

[16] Prasad, T. S. K. et al. (2009) Human Protein Reference Database 2009 Update. Nucleic Acids Research. 37, D767-72.

[17] Quinlan, J.R. (1993) C4.5: Programs for machine learning. Morgan Kaufmann, San Mateo, CA.

[18] Revesz, P. Introduction to Databases: From Biological to Spatio-Temporal, Springer, New York, 2010.

[19] Revesz, P., Triplet, T. Classification Integration and Reclassification using Constraint Databases, Artificial Intelligence in Medicine, vol. 49, no. 2, pp. 79-91, 2010.

[20] Revesz, P., Triplet, T. Temporal Data Classification using Linear Classifiers, Information Systems, vol. 36, no. 1, pp. 30-41, 2011.

[21] Shortridge, M., Triplet, T., Revesz, P., Griep, M., Powers, R. Bacterial Protein Structures Reveal Phylum Dependent Divergence, Computational Biology and Chemistry, vol. 35, no. 1, pp. 24-33, 2011.

[22] Shen, J., Person, M. D., Zhu, J., Abbruzzese, J. L., Li, D. Protein expression profiles in pancreatic adenocarcinoma compared with normal pancreatic tissue and tissue affected by pancreatitis as detected by two-dimensional gel electrophoresis and mass spectrometry. Cancer Research, vol. 64, pp. 9018-26, 2004.

[23] Thuraisingham, B.: A Primer for Understanding and Applying Data Mining, IT Professional, pp. 28-31, 2000.

[24] Triplet, T. Shortridge, M., Griep, M., Stark, J., Powers, R., Revesz, P. PROFESS: a PROtein Function, Evolution, Structure and Sequence database, Database – The Journal of Biological Databases and Curation, doi no. 10.1093/baq011, 2010.

[25] UniProt from *http : //www.uniprot.org/*

[26] WEKA - Machine Learning Group at University of Waikato from *http : //www.cs.waikato.ac.nz/ml/*

[27] What is a contingency table? from *http : //www.eumetcal.org*

[28] Witten, I. H. and Frank, E. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco, 2 edition, 2005.

[29] Yamada, M., Fujii, K., Koyama, K., Hirohashi, S., Kondo, T. The Proteomic Profile of Pancreatic Cancer Cell Lines Corresponding to Carcinogenesis and Metastasis. Journal of Proteomics & Bioinformatics, vol. 2, pp. 18, 2009.