PREDICTING eGFR TRENDS OF HEALTHCARE RECORDS

COMM033 – Database and Data Mining Coursework

Giang Tran-Hoang
Department of Computing, FEPS
University of Surrey, Guildford, Surrey, UK

Yan Huang
Department of Computing, FEPS
University of Surrey

Di Wang
Department of Computing, FEPS
University of Surrey

***Abstract*—Healthcare data are usually very sparse and noisy and there are so many potential information yet to be found. This project is an attempt to apply data mining techniques to predict the eGFR variation trend of a patient in a period of time, given his examination results.**

Keywords—eGFR, bag of words, Naïve Bayes, regression

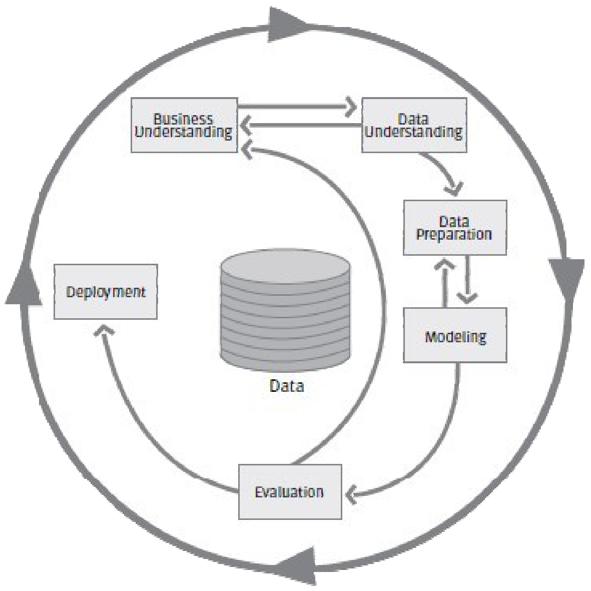
I. INTRODUCTION

Healthcare database usually comes in the form of tables of examination results. For a patient, each of his examination results will be saved as a record in the database. This way, the number of data items for one patient is vast and can only be retrieved by database queries that will take a long time to run. Moreover, the number of as well as the exact examinations taken by the patient are quite different, and we have no data about that patient regarding that feature if he does not take the health service. That will make the actual information in such database very sparse and potentially incorrect or incomplete.

With that characteristic of healthcare data in mind, our group chose eGFR - estimated glomerular filtration rate [2] value as the target because this code appears in almost all patients' data. The task is to predict whether this value is going to increase or decrease over a period of time based on all the other examination results that patient has been taking during the same time frame. We follow the CRISP-DM (Cross Industry Standard Process for Data Mining)[2] framework to transformed the collected database into usable data format, then build prediction models using two of the most classic but powerful classifier: Naïve Bayesian and Logistic Regression. The data will be transformed into bag-of-word presentation, and then vectorized. The data vectors acquired will be used as the training and testing set for those two filters. The output model not only acts as a classifier, but also provides information about which examination code has the most impact to the development of eGFR value through the coefficient matrix. We will then evaluate the improved classifier where all the low-impact features remove from the data.

II. THE CRISP-DM FRAMEWORK

This framework is proposed and maintained by SPSS and several high-profile data mining firms [1]. The framework suggests 6 phases for any data mining project:



- Business Understanding: specify goals, objectives and related business needs. Draw a problem definition and preliminary plan based on this knowledge.
- Data Understanding: be familiar with the data, point out problems, interesting sub-set or hypothesis.
- Data preparation: from raw data, many activities would be applied in order to clean, select, transform that into final input of the models.
- Modeling: build the model and optimize the parameters
- Evaluation: test the models thoroughly and review the processes. Compared with predefined goals and make changes if needed.
- Deployment: reorganizing, packaging and presenting the results into some format that users can use the most easily. Then deliver the system.

In this particular project, the part of data collection and some of data preparation was already done for us. For other parts, we will strictly follow the framework, and those steps will be reflected later in the report.

III. BUSINESS UNDERSTANDING AND DATA UNDERSTANDING

A. The goal of the project:

This project has a small difference from other data mining projects. The healthcare data have already been collected before we need to choose what to do with it. But this is actually reasonable for this type of data where the information is sparse and potentially incorrect of incomplete, while there could be so many hidden patterns and relationships between the features. This database centers around diabetes deceases so the choice being made was to target the eGFR, a measurement directly indicate the strength of the kidney [1]. This code is very common among patients in this database, making sure that we have enough data for training and validating the model. During a period of time, this value will change according to many factors and its movement is one of the biggest concerns of the doctor. Therefore predicting the trend of this crucial factor will be the main aim of our efforts here. The plan is to use the classifier with possibility output, which will allow us to trace back to the model and see which factors are more impactful to the output. This is the other objective that we are trying to reach as well.

B. The characteristics of the data and our approach:

The whole project will follow the CRISP-DM framework, so our first task after deciding on the goal is to investigate on the data. The database has five tables, with the main data source for the project is journalsubset. Each row of this table consists of the patient's ID, the code (drug, treatment, decease, measurement,…), the date and the values (if available). The code and its corresponding date are the main valuable information in this database. The values of some code can contain information, but because the majority of the codes are non-numerical, and the domains of the codes with value are different, simple using those values are not easy. With this analysis, our decision is to use the bag-of-word presentation for the patient record. The final data model will be for each of patient, the trend of eGFR value (up or down) during a period of time will be the output, and the appearance of other codes during the same time frame will be the input.

Further investigation gave us some improvements on the data model. We found out that the codes in the journalsubset table are like expansions of the codes in the codinggroup table. Each of those will start with one code in codinggroup as the prefix, and are actually just variations of the prefix code, mostly different ways to measure one same feature, E.G: measuring blood pressure at home or at the clinic. Therefore, to reduce the vocabulary size of the bags of words, we decided to treat all the codes start with the same prefix as just one code represent by the prefix code itself. This will reduce the vocabulary size from over 11000 to just over 1400 maximum without losing much information. Another big decision is that for all the codes that have values, to make use of both the code and its values, we evaluate its values during the time-frame and check if it goes up or down. Then instead of just putting the code name as a word into the bag, we put the word <code>_up or <code>_down instead. This way, we capture some meaningful information into a format that compatible with the presentation of choice.

IV. DATA PREPARATION:

A. Getting the eGFR variations as ground truth:

The first task is to extract the eGFR values for all the patient ID and find out the trend during each period of time. The first decision to make is the length of the time frame. Because we only capture the movement of the values as linear level, which means that if the curve of the value during that time is more complex, some information can be lost. Setting the frame to small values can help avoiding that, but this way the number of records increases quite a bit. Furthermore, there are many periods of time where the patient did not have any examination, and this creates gaps in the data domain which may cause loss of information. So there is a trade-off between the information loss, performance and complexity that we had to balance. After testing with many values, we choose to use the time frame of one and half year, which will cover almost all the time span for each patient. To get the ground truth which is the eGFR trends, for each patient, we choose a start date with eGFR value, then find the latest date within the time frame and calculate the different. Positive means going up, and we label it as 1, non-positive trends are labeled as 0. Then the end date of the previous frame will be the start date for the next frame and we continue until there is no record left for that patient and move to the next one.

B. Building the bags of words and vectorizing them:

After querying to get the ground truth as described above, we save the data into a CSV files with the format of each line is <patient-ID, start-date, end-date, variation>. Using this as the input for the next query, which is for building the bags of words for each of those records. For each of the patient, the query gets all the codes that are not eGFR, then matches those whose date is in the time frame of the record and saving the code into the text presentation of that patient time window. The transformation of codes with value into *_up and *_down format is done using dictionary, saving all the examination results within the period, and getting the different between the latest and the earliest. Note that this kind of code again only has value of linear grade, capturing the general trend of increasing or decreasing, no more details of the curve is saved.

After each patient record in a time frame has been assigned a text string (bags of word), we used the vectorizers of the sklearn [3] python library. This will take in all the text strings and turn them into vectors of word counts. The number of dimensions is the total vocabulary size of the set, and the value in each element of a vector will be the number of appearance of that code in the time frame. We also test the tf-idf [4] measurement as an alternative for just counting. This is a metric that takes the impact of a word towards the whole dataset into account instead of just towards the document that word is found. The calculation for tf-idf is the product of two part:

- tf (term frequecy) = count(word)/count(all) – the ratio of the number of appearance of that word over the total number of words in that string.
- idf (inverse document frequency) = log(n/n(word)) - with n is the total number of text strings and n(word) is the number of strings with that word in it

This metric will eliminate the impact of the codes that appear in almost all the records which will have little discrimination effect. With the output of this phase is the set of vectors representing the information of

V. MINING MODELS:

To build the classifier, we choose two of the most basic but powerful techniques: the Naïve Bayesian (NB) filter and logistic regression. The data is labeled as 0 and 1 already, so these two can be directly applied with little change. The actual implementation makes use of the sklearn library again: MultinomialNB class for NB filter and linear_model class for logistic regression. This part of the report will discuss why we chose those two and how we used them.

A. Naïve Bayesian Filter:

The NB is a simple classifier with probability output and is very easy to build and fast to run [5]. The theory behind this is to assuming that each feature is independent to other. This way, when the output of the model can be interpret as a probability metric: $p(class=C|F_1,\dots,F_n) = = \frac{p(C)p(F_1,\dots,F_n|C)}{p(F_1,\dots,F_n)}$ where $p(F_1,\dots,F_n|C) = p(F_1|C)p(F_2|C)\dots p(F_n|C)$ due to the assumption above. All of the required value to get the prediction for the class of a specific set of feature can be calculated easily using simple statistic, and the training only need to run once for a dataset. This characteristic makes NB such a fast tool, especially when the dimension size is so large in like in this scenario. Another reason why NB is suitable for this problem is that the data values in vectors are quite sparse but discrete and have low domain size. This will make the values of $P(F_i)$ more meaningful that the case where those features are continuous and has too many possible values.

The implementation of this filter in python is very straight forward with a simple call:

```
classifier = MultinomialNB()
classifier.fit(training_feature_matrix,label)
```

These will output a trained NB classifier that we can directly use to predict the output of an input vector (using `classifier.predict(input_vector)`) or evaluate the performance against a labeled testing data (`classifier.score(testing_set)`).

B. Logistic Regression:

Regression, in this scenario is logistic regression as the output is probability values between 0 and 1, is another simple but powerful techniques for classification [6]. This technique involves building and training a multiplication matrix that can map a vector of input values into a continuous output values. The function being used in this logistic filter is:

$$f(t) = \frac{1}{1 + e^{-t}}$$

This function will make sure the output of the model is a value between 0 and 1. This feature makes logistic regression very suitable of a problem with only two possible outputs (eGFR going up or down) like in this project. Another valuable output of this type of model is the weight matrix, which will denote which features have more impacts on the output than the other. This will be great findings as well as hint for improvement of the model itself.

Again the implementation of the model in python is very simple with just several calls:

```
classifier = MultinomialNB()
classifier.fit(training_feature_matrix,label)
```

The usage of the trained classifier is the same as NB above.

C. Validating the Model:

To validate the model, we choose to use the repeated random sampling cross-validation techniques [7]. The whole data set will be randomly split to two parts: one of training and one for testing with the ratio of 19:1. The trained model using training set then will be tested on the testing set and the average accuracy will be recorded. We repeat the process 10 times and get the final evaluation as the average accuracy of all 10 loops.

This kind of classification problem with bag-of-word presentation depends on the size of the training data, so the split of 19:1 give more date for training which will increase validation correctness. Moreover, random sampling with 10 loops will make sure all the data is covered in a fair way and the model will be tested throughout. This validation process is the best for this scenario.

VI. EVALUATION AND SUGGESTION FOR DEPLOYMENT:

A. Validation result:

Using the CountVectorizer, the accuracy for NB and Logistic Regression is 62.93% and 61.26% respectively. This is a bit lower that what we expected, but still acceptable. Another attempt with tf-idf metric gives the results of 63.21% and 60.13%. The slightly poor performance is probably caused by the fact that we capture too little information from all the code with values including the ground truth. Only the most generic trend is recorded, the actual curve and the absolute values of the differences are not used as all.

B. Improvement of the classifier:

After using two classifiers with the result just over the bar, we tried to improve the classifier by removing low impact codes. From the output model of logistic-regression, we can have the coefficient vector of the words in the vocabulary. We then remove all the words with coefficient absolute value lower than a certain threshold and run the classifier again. After testing with several threshold values, we got the best improvement with value of 0.33: accuracy of 68.89% and 65.38% for NB and logistic regression respectively. With this result, we can conclude that even it seems that we cut out some information by chopping the vocabulary, we actually cleaning up the data and make better classifier from that. Those codes which are removed can be considered as noisy of incorrect data because they are bringing the classification performance down despite holding information.

C. Suggestion for future improvements and deployments:

Even after cleaning up the data, the accuracy is still not impressive enough. This may be caused by the lack of information in the data presentation or the inconsistency of the data itself. For the bag of word presentation, many patient windows have a lot of code, but still a fair number of them have only several codes. Those data items have little information value but have the same affection to the classification result as other items with a lot of information do. Our guess is that if we remove those data item with too little vocabulary size, we will make better classifier. That will be our first attempt to improve the result if we have more time.

The other potential cause which is missing information in the encoded bags of word can be solved by capturing more information and putting more codes into the vocabulary. The codes with values only appear in the data presentation in the form of up of down trends. There is nothing about the magnitude of the difference or the turning points of the curve during the window retained in the data model. If possible, we should encode all those info into new codes if possible. For example, if code A during a window goes up very steeply and then goes down slowly, we can have the code A_upup_dow, and if during another window code A goes down slowly and then goes up slowly, followed by another steep uphill move, we will have the code A_down_up_upup. This way, both the intensity and direction of the trend are captured throughout the window, not just the start and end.

Another suggested improvement is to change the way we verify the model. This classifier aims to predict eGFR trend of unknown data records, so instead of choosing randomly, the test set should either be of different patients from those of the training set or be of the windows in later dates. This way, we will make the testing closer to the real usage of the classifier and give more confidence to user of the system.

VII. CONCLUSION:

The project was a great learning experience. We had real-world data to test on and dealt with realistic problems, we explored and practiced all processes of data mining using a structural methodology and reached our main goals with many options still to try on. The bag of word presentation is quite a clever way to extract information from this type of database, but the level of feature extraction in this project is still quite shallow and much of the information still missed out. Further efforts to encode more information will be a big boost to the evaluation of the model. Both of two classifiers being used, Naïve Bayes and Logistic Regression, performed considerably well and produced similar results. Using those two fast techniques, the model actually requires more time to extract data than building and training itself. However this fact can change if we use other classifiers such as neural network because the dimension of the data is massive. In conclusion, the project is a success with many achievements for a very challenging task, but there is still a lot of room to improve.

REFERENCES

[1] Patient.co.uk "Definition of Estimated Glomerular Filtration Rate", available online at: http://www.patient.co.uk/health/Estimated-Glomerular-Filtration-Rate.htm (last visit 20/05/2013)

[2] P. Chapman et al "CRISP-DM Step by step data mining guide", SPSS documentation for CRISP-DM 1.0, 2000

[3] Pedregosa *et al.*, Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011

[4] K.S Jones "A statistical interpretation of term specificity and its application in retrieval". *Journal of Documentation* **28** (1): 11–21 1972

[5] Process Software "Introduction to Bayesian Filtering - Using Bayes' Filtering to keep the spam out of your inbox". Available: http://www.process.com/precisemail/BayesianFiltering.pdf

[6] D. Hosmer, S. Lemeshow "Applied Logistic Regression" (2nd ed.). Wiley. ISBN 0-471-35632-8, 2000

[7] R. Hirsch "Validation Samples". Biometrics 47 (3): pp: 1193–1194, 1991

CONTRIBUTION:

Giang Tran-Hoang: write the python program for bag-of-word building, write the report.

Di-Wang: write the python program for eGFR trend extraction, and the reused MySQL connecting code. Prepare the presentation.

Yan-Huang: write the python program for building model, testing models, extracting important and less impactful features for improved classifier. Provide test results for report and presentation.

Above are just the tasks that each person was individually responsible for. Three members worked very close together and all important decisions on the objectives, methodologies, techniques used in this project are agreed upon thorough discussions. We also help each other with the individual tasks: help debugging code, suggest code improvement for faster executing, proofreading and editing the presentation and report.

APPENDIX

A. *Get Time Frame*

```
    for i in ids:
        count+=1
        c.execute ("SELECT * FROM qickd2.journalsubset where
code like 'eGFRorig' and id like '{}'".format(i))
        person=c.fetchall()
        tmp=person[0][2]
        for j in range(len(person)):

            for k in range(j,len(person)):
                if                              ((person[k][2]-
person[j][2])>datetime.timedelta(days=182))and          (tmp<
person[k-1][2])             and            ((person[k-1][2]-
person[j][2])<datetime.timedelta(days=182))  and  (person[k-
1][2]!=person[j][2]) and(person[j][2]>=tmp):

csvfile.writerow([person[j][0],person[j][2],person[k-
1][2],person[k-1][3]-person[j][3]])
                   tmp=person[k-1][2]
                   break
```

B. *Get the Codes*

```
    for i in csv_file:
        if not (i['ID'].startswith(currentID)):
            file1.flush()
            currentID = i['ID']
            c.execute  ("SELECT  *  FROM  qickd2.journalsubset
WHERE   code   not   like   'eGFRorig'   and   id   like
'{}'".format(currentID))
            currentcode = list(c.fetchall())
        if len(currentcode)==0:
            continue

file1.write(i['ID']+':'+i['start_date']+',,,'+i['variation']
+':')
        valued_code={}
        for code in currentcode:
            if   str(code[2])   >=   i['start_date']   and
str(code[2]) <= i['end_date']:
                if code[3] == 0.00:
                    for prefix in code1:
                        if code[1].startswith(prefix):
                            file1.write(prefix+' ')
                            break
                else:
                    for prefix in code2:
                        if code[1].startswith(prefix):
                            if valued_code.has_key(prefix):

valued_code[prefix].append([code[2],code[3]])
                            else:
```

```
valued_code.update({prefix:[[code[2],code[3]]]})
                            break
                currentcode.remove(code)
        for prefix in valued_code.keys():
            if len(valued_code[prefix]) <= 1:
                continue
            else:
                max_val = valued_code[prefix][0]
                min_val = valued_code[prefix][0]
                for val in valued_code[prefix]:
                    if max_val[0] < val[0]:
                        max_val = val
                    if min_val[0] > val[0]:
                        min_val = val
                if max_val[1] - min_val[1] > 0:
                    file1.write(prefix+'_up ')
                else:
                    file1.write(prefix+'_down ')
```

C. Build Vector_count

```
    for line in f:
        key,variation=line.split(',,,')
        variation=float(variation.split(':')[0])
        p=np.random.random()
        if p>prob:
            sample+=1
            y.append(variation)
            if variation>0:
                label.append(1)
            else:
                label.append(0)
            code.append(line.split(':')[2].strip().split('
'))
            words=line.split(':')[2].strip().split(' ')
            for word in words:
                if cmp(word,'')!=0:
                    if not feature_names.has_key(word):
                        feature_names.update( {word:n} )
                        n+=1
        else:
            sample_test+=1
            y_test.append(variation)
            if variation>0:
                label_test.append(1)
            else:
                label_test.append(0)

code_test.append(line.split(':')[2].strip().split(' '))


    print sample
    vector=np.zeros((sample, n))
    for i in range(sample):
        for word in code[i]:
            if cmp(word,'')!=0:
                index=feature_names[word]
                vector[i][index]+=1
```

D. Build Vector_TFIDF

```
         for line in f:
            key,variation=line.split(',,,')
            variation=float(variation.split(':')[0])
            p=np.random.random()
            if p>prob:

code.append(line.split(':')[2].strip().replace('/',''))
                d.update( {key:n} )
                y.append(variation)
                if variation>0:
                    label.append(1)
                else:
                    label.append(0)
                n+=1
            else:

code_test.append(line.split(':')[2].strip().replace('/',''))
                y_test.append(variation)
                if variation>0:
                    label_test.append(1)
                else:
                    label_test.append(0)
                n+=1
```

```
        tfidf                                                   =
CountVectorizer(ngram_range=(1,1),token_pattern=ur'\b\w+\b',
lowercase=False)
        training_feature_matrix =tfidf.fit_transform(code)
        testing_feature_matrix =tfidf.transform(code_test)
```

E. Get Most Important Features

```
classifier = MultinomialNB()
        classifier.fit(training_feature_matrix,label)
        coef=[]
        for n,i in enumerate(classifier.coef_):
            coef.append( [i,n] )
        coef=sorted(coef,reverse=True)[0:5]
        for i in coef:
            print tfidf.get_feature_names()[i[1]]
```

F. Get Imporved Code

```
feature_names=tfidf.get_feature_names()
        vocal={}
        for i in range(len(feature_names)):
            vocal.update(   {feature_names[i]:coefficient[i]}
)
        new_code=[]
        for s in code:
            words=s.split(' ')
            temp_str=''
            for word in words:
                try:
                    if abs(vocal[word])>=threshold:
                        temp_str+=word+' '
                except:
                    temp_str+=word+' '
                    continue
            temp_str=temp_str.strip()
            new_code.append(temp_str)
```