2020

COMP723 Data Mining and Knowledge Engineering

Millan Uka 17981567

Jasper Uy 17982348

Part A
https://www.researchgate.net/publication/221178905_Data_Mining_in_Healthcare_Information_Systems_Case_Study_of_a_Veterans_Administration_Spinal_Cord_Injury_Population/link/00b7d5304b166ea9ab000000/download

Background

The organization that initiated this data mining application was the Veteran Health Administration (VHA). With help from researchers from the University of Illinois and the University of Loyola University. The VHA, as the name suggest focus on providing healthcare to U.S. veterans. They have 1255 healthcare facilities, 170 of which are medical centres. The researchers in this case study focuses on one medical centre, the VHA hospital located in Chicago. This hospital uses the Veterans Health Information Systems and Technology Architecture (VistA). This integrated system, contains the SCI database, which important in this case study. SCI (Spinal Cord Injuries) patients have long lengths of stays (LOS) and therefore large costs. This means that a model that can predict LOS is incredibly important, in-order help reduce the costs of SCI care.

Target Application

The model that the researchers developed was intended to reduce the costs of caring for SCI patients. SCI patient usually have longer stays and increased costs compared to other patients. The model uses nursing diagnosis to predict the length of stay. Knowing the length of stay for a patient is incredibly important in determining the allocation of resources. Proper allocation of resources leads to reduced costs of SCI care.

Data Description

Within the Hospital Information Systems, the data mainly is used for the purpose of knowledge discovery through history records from old to new operations, how this assist the hospital is through the analysis of searching patterns with the raw data that they produced from their vast size of databases. The count of the data is distributed into 20 diagnostic categories, in those contain 4,750 nursing diagnoses that are cumulated within an 11-year database, with further distribution, there are 161 unique nursing diagnoses.

The significant categories taken into consideration are Skin Care, Elimination, Self-Care Deficit, Infection Prevention/Control, Mobility, Respiratory Function, Psychosocial Adaptation, Community Reintegration, Pain Management, Knowledge Deficit, Nutrition, Fluid Volume Maintenance, Acute Problem Management, Safety from Injury, Rest, Cognitive Functioning, Temperature Control, Sexual Health, Communication and Miscellaneous. These categories are mainly chosen due to their repetition appearance, whilst the other data of categories are listed under Miscellaneous with their lack of data of not appearing of more than 11 times during the 11-year study period.

The 20 of the nursing diagnoses categories became the primary of input variable for the investigation of data to assist the development of ANN and assisting with the performance of prediction of LOS.

Mining tools

This data mining application several different tools and algorithms. The researchers for this case study use Knowledge discovery as their framework. Knowledge discovery begins with data acquisition in which data is collected, this data is then cleaned and validated. The data is then aggregated and visualized. The data will then have various algorithms applied to mine the data and extract patterns within in the data.

The first phase is the data acquisition phase, in this phase, data is collected and extracted from its source. In this case it is from an ORACLE database from VistA. The patient admissions are extracted.

There was a total of 1107 admissions with 597 patients. The admissions were then used to find the nursing diagnosis for each admission.

The second phase is data validation and cleaning phase. A lot of the data collected from the database was non-standardized. The data was visually inspected to check for any noise in the data. Values that were thought to be random were deleted. Not in registry values were added to values that were collected earlier in the study when the registry was not available. Admissions from employees of the hospital were removed. Diagnoses of Multiple Sclerosis and Gullian Barre Syndrome were removed. At the end of this phase there were 1107 admissions with 525 patients.

In data aggregation step. Data was clustered together into 20 categories labels. The labels were checked by two domain experts. The next step was data visualization. The length of stay values were all normalized. In this phase by visualizing the data the researchers managed to determine that skin care, elimination, self-care deficit, infection prevention/control, and respiratory function were the categories with the highest length of stay.

After visualizing the data, various algorithms were applied to the data. Four different types of Artificial Neural Networks (ANN) were used. Dynamic network, prune network, radial basis function (RBF) network, multilayer perceptron (MLP). These ANNs have the same general architecture. An ANN usually has three or more layers. One input layer, one or more hidden layers and one output layer. Each node in the architecture is called a neuron. The connections between the nodes in the different layers have weights which are adjusted during training of the ANN. In the input layer there is 20 neurons one for every category of diagnosis. The output layer has 1 neuron for LOS. The hidden layers varied between the different types. Dynamic used two hidden layers with 3 in the first and 5 in the second. MLP uses two hidden layers with 27 in the first and 2 in the second. The RBF has 1 hidden layer with 16 neurons. The Prune ANN has 1 hidden layer with 20 neurons. Each ANN was run 800 times. The accuracy scores and mean squared error (MSE) were collected. Over the 800 iterations the model with lowest out-of-sample MSE was selected as the best model the model selected was the RBF ANN.

Discussion

The goal of this research was to predict the length of stay for SCI patient at the VHA. The most optimal model that they found for predicting LOS was the RBF ANN. The RBF ANN got an accuracy score of 77.58% which is lower than Dynamic, MLP and Prune, which got 77.94%, 78.00% and 78.34% respectively. However, it was selected due to the lowest out-of-sample MSE of 0.069616426. According to this study, the direct and costs of SCI cases are estimated to be between $7.2bn and $9.5bn. The average cost for first year SCI care is $223,000USD, with an additional cost of $26,000USD every year. This model can accurately predict 77.58% of SCI cases' LOS. This would help to reduce a significant amount about of the estimated $7.2bn-$9.5bn. It would be difficult to determine an exact savings of implementing this model in the VHA without further research. However, I think generally they managed to achieve their goals of accurately predicting LOS and therefore their goal of reducing costs. However, I think there were several issues with the model used in the paper. The first issue was that nursing diagnosis are not the only variable that affect the LOS of SCI patients. None of the diagnosis categories were found to be significant factor in determining LOS, this is because of the various variables that affect LOS. Other variables that affect LOS are, financial, psychology and social issues that SCI patients experience. There is also a lack of community long term care meaning that SCI patients have to stay at the VHA longer. Another issue was that the nursing diagnosis were not standardized which meant that there had to be a lot time spent on visual inspection. There could have been also been context lost when the data was standardized. Overall, I think the researchers managed to achieve their goals. However, there is some areas that they could improve with more research.

Part B

Question 1

```
from sklearn.model_selection import train_test_split
import pandas as pd

# Read the data into a pandas dataframe
data = pd.read_csv('pima-indians-diabetes.data.csv',delimiter=',')

#Remove the target variables for x. Y only has the target variables
x = data.drop("Class variable (0 or 1)", axis=1)
y = data["Class variable (0 or 1)"]
#Split it. Training has 70% of the records. Test has 30%
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

#Combine the target and features together
train = pd.concat([X_train, y_train], axis=1)
test = pd.concat([X_test, y_test], axis=1)

#save the two sets to csv files
train.to_csv("train.csv")
test.to_csv("test.csv")
```

The csv files were downloaded formatted so that they have appropriate columns name and the split into a test and train csv file. This is done to keep the data consistent, and therefore ours results comparable. Code shown above

Step 1: (4 marks)

Use the decision tree classifier and tune the max_depth parameter. Often the default setting of 2 produces poor results due to overfitting. Vary the max_depth parameter in the range [2, 20] in steps of 2 and record the classification accuracy on the test set for each value of this parameter.

Generate a two-column table (max_depth, test accuracy).

This code snippet below. Goes from 2-20 in steps of 2. With the max depth of the decision tree being the I variable which changes during every iteration.

```
from sklearn.tree import DecisionTreeClassifier
for i in range(2, 20, 2):
    classifier = DecisionTreeClassifier(max_depth=i)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    print(str(i) + " " + str(accuracy_score(y_test, y_pred)))
```

max_depth	Test accuracy
2	0.696969696969697
4	0.6926406926406926
6	0.6623376623376623

8	0.670995670995671
10	0.6753246753246753
12	0.658008658008658
14	0.6666666666666666
16	0.6623376623376623
18	0.653679653679653

Step 2 (4 marks)

Now use the neural network (MLP) classifier and perform tuning on the learning rate parameter. Vary the learning rate in the range of [0.001, 0.01] in increments of 0.001 and record the classification accuracy on the test set for each value of this parameter.

Generate a two-column table (learning rate, test accuracy).

The code snippet below iterates through a list of numbers ranging from 0.001 to 0.01 in steps of 0.001. In then classifies them using the MLP classifier. During this step I did get warnings about early convergence of the model

```
from sklearn.neural_network import MLPClassifier
x = np.arange(0.001, 0.01, 0.001)
for i in x:
    classifier = MLPClassifier(learning_rate_init=i)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    print(str(i) + " " + str(accuracy_score(y_test, y_pred)))
```

Learning rate	Test accuracy
0.001	0.7272727272727273
0.003	0.7142857142857143
0.004	0.7402597402597403
0.005	0.7056277056277056
0.006	0.7186147186147186
0.007	0.7056277056277056
0.008	0.70995670995671
0.009	0.683982683982684

Step 3 (5 marks)

Use the best value of the max_depth parameter from step 1 and perform feature selection using Python's *SelectKBest()* method with the decision tree classifier. Vary the K parameter in the range [2..7] in increments of 1 and evaluate the accuracy on the test set for each value of K.

Generate a two column table (K, test accuracy).

The code snippet below goes from 2 to 7 in steps of 1. It uses SelectKBest function to select the best feature and creates a model, and prints out the accuracy score.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
for k in range(2, 7, 1):
    fs = SelectKBest(chi2, k=k)
    fs.fit(X_train, y_train)
    X_train_fs = fs.transform(X_train)
    X_test_fs = fs.transform(X_test)
    classifier = DecisionTreeClassifier(max_depth=2)
    classifier.fit(X_train_fs, y_train)
    y_pred = classifier.predict(X_test_fs)
    print(str(k) + " " + str(accuracy_score(y_test, y_pred)))
```

k	Test accuracy
2	0.6926406926406926
3	0.6883116883116883
4	0.7142857142857143
5	0.696969696969697
6	0.6623376623376623

Step 4 (5 marks)

Use the best value of the learning rate parameter from step 2 and perform feature selection using Python's SelectKBest() method with the MLP classifier. Vary the K parameter in the range [2..7] in increments of 1 and evaluate the accuracy on the test set for each value of the learning rate.

Generate a two-column table (K, test accuracy).

Same thing as the previous step, but instead with the MLP classifier

```
for k in range(2, 7, 1):
    fs = SelectKBest(chi2, k=k)
    fs.fit(X_train, y_train)
    X_train_fs = fs.transform(X_train)
    X_test_fs = fs.transform(X_test)
    classifier = MLPClassifier(learning_rate_init=0.004)
    classifier.fit(X_train_fs, y_train)
    y_pred = classifier.predict(X_test_fs)
    print(str(k) + " " + str(accuracy_score(y_test, y_pred)))
```

k	Test accuracy
2	0.7402597402597403
3	0.7359307359307359
4	0.7359307359307359
5	0.7402597402597403
6	0.7489177489177489

Step 5 (12 marks)

In this step we will combine the two classifiers into a single classifier. To do this first look up the *sklearn* documentation on the use of the *predict_proba()* method that returns a vector of probabilities for each class for a given test sample. For the diabetes dataset, this will return a vector of size 2 as there are two classes.

a) For each test sample apply *predict_proba()* for the decision tree model produced from step 3 and select the class that gives the highest probability value.
b) Repeat this process for the MLP model produced from step 4.

The two models are combined by classifying each test instance into the class that produced the highest probability from steps 5 a) and 5 b).

Generate the average classification accuracy on the test set that is produced by combining the two classifiers.

The code snippet shows the initialization of the two classifiers. It also shows the probabilities being calculated with the predict_prob function. They are also averaged

```
# initiate the Decision tree classifer using the best parameters determined in previous steps
classifier = DecisionTreeClassifier(max_depth=2)
fs = SelectKBest(chi2, k=4)
fs.fit(X_train, y_train)
X_train_fs = fs.transform(X_train)
X_test_fs = fs.transform(X_test)
classifier.fit(X_train_fs, y_train)
# calculate the probabilties for the decision tree classifier
dt_prob = classifier.predict_proba(X_test_fs)

# Do the same for the MLP classifier
classifier = MLPClassifier(learning_rate_init=0.004)
fs = SelectKBest(chi2, k=6)
fs.fit(X_train, y_train)
X_train_fs = fs.transform(X_train)
X_test_fs = fs.transform(X_test)
classifier.fit(X_train_fs, y_train)

# calculate the probabilties for the decision tree classifier
mlp_prob = classifier.predict_proba(X_test_fs)

# Average the probablities for both classifiers
probas = (dt_prob + mlp_prob) / 2
```

The code snippet below is a continuation of the code snippet above. This code goes through all averaged probabilities it then classifies them by checking the probability. The accuracy is then outputted.

```
y_pred = []
# Go through every probability and determine the higher probability of the class
for i in range(len(probas)):
    # if probability is aboive 0.5 then diabetes is not present
    if probas[i][0] > 0.5:
        y_pred.append(0)
    # diabetes is present
    else:
        y_pred.append(1)

print(accuracy_score(y_pred, y_test))
```

The output that we got when combining both the classifiers was 0.7402597402597403. This is much higher than any values in the decision tree results. However, it is not better than the best result in the MLP classifier results of 0.7489177489177489.

Question 2 ***(Total of 40 marks)***

1) Use the *sklearn.MLPClassifier* with default values for parameters and a single hidden layer with k=20 neurons. Use default values for all parameters other than the number of iterations which should be set to 150. Also, as is standard for an MLP classifier, we will assume a fully connected topology, that is, every neuron in a layer is connected to every other neuron in the next layer.

   Record the classification accuracy as it will be used as a baseline for comparison in later parts of this question. **(5 marks)**

This code snippet is uses an MLP model with one layer with 20 neurons. Which is used as the baseline for the rest.

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(hidden_layer_sizes=(20), max_iter=150)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(str(accuracy_score(y_test, y_pred)))
```

The output was 0.683982683982684

2) We will now experiment with two hidden layers and experimentally determine the split of the number of neurons across each of the two layers that gives the highest classification accuracy. In part 1 of the question we had all k neurons in a single layer. In this part we will transfer neurons from the first hidden layer to the second iteratively in step size of 1. Thus for example in the first iteration, the first hidden layer will have k-1 neurons whilst the second layer will have 1, in the second iteration k-2 neurons will be in the first layer with 2 in the second and so on. Summarise your classification accuracy results in a 20 by 2 table with the first

column specifying the combination of neurons used (e.g. 17, 3) and the second column specifying the classification accuracy. **(7 marks)**

The code below iterates between 1-20 and transfer the neurons from the first to the second layer.

```
for i in range(1, 20, 1):
    classifier = MLPClassifier(hidden_layer_sizes=(20-i, i), max_iter=150)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    print("(" + str(20-i)  + ", " + str(i) + ") " + str(accuracy_score(y_test, y_pred)))
```

Neuron combination	Test accuracy
(20)	0.683982683982684
(19, 1)	0.645021645021645
(18, 2)	0.7186147186147186
(17, 3)	0.7272727272727273
(16, 4)	0.6493506493506493
(15, 5)	0.7229437229437229
(14, 6)	0.7272727272727273
(13, 7)	0.7316017316017316
(12, 8)	0.7489177489177489
(11, 9)	0.7186147186147186
(10, 10)	0.7229437229437229
(9, 11)	0.7359307359307359
(8, 12)	0.7532467532467533
(7, 13)	0.7229437229437229
(6, 14)	0.7575757575757576
(5, 15)	0.7359307359307359
(4, 16)	0.7402597402597403
(3, 17)	0.7229437229437229
(2, 18)	0.645021645021645
(1, 19)	0.645021645021645

3) From the table created in part 2 of this question you will observe a variation in accuracy with the split of neurons across the two layers. Give explanations for some possible reasons for this variation. **(8 marks)**

The lowest accuracy score was 0.64502, there were three combinations that had this. They were (19, 1), (2, 18) and (1, 19). These were all combination with heavily imbalanced combinations. This could mean that a possible reason for them getting lower accuracy scores was because of the imbalance of neurons between hidden layers. The highest accuracy score was 0.7575 with a combination of (6, 14). The difference between the first combination, (20) and the next combination, (19, 1) is interesting. This could have been caused by the increased complexity that another hidden layer as well as the imbalance of neurons. The median is 0.7229437 and variance is low at 0.001403104. The IQR is also relatively low with 0.02597403. This indicates other than some of the imbalanced combinations the variation between values is not incredibly different from each other.

4) By now you must be curious to see whether the trends that you noted in step 3 above hold true for other datasets as well. Use the car evaluation dataset from

(https://archive.ics.uci.edu/ml/datasets/Car+Evaluation) to carry out the same experiment and generate the same table as in part 2 of this question. Comment on your results from this section. **(10 marks)**

This uses similar code to the step 2. However, label encoding is applied to the car data columns to convert the categorical data to numerical data.

```
data = pd.read_csv('car.data.csv',delimiter=',')

from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in data.columns:
    data[i]=le.fit_transform(data[i])

print(data)
x = data.drop("class", axis=1)
y = data["class"]
#Split it. Training has 70% of the records. Test has 30%
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

# initialize the baseline classifer
classifier = MLPClassifier(hidden_layer_sizes=(20), max_iter=150)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print(str(accuracy_score(y_test, y_pred)))

for i in range(1, 20, 1):
    classifier = MLPClassifier(hidden_layer_sizes=(20-i, i), max_iter=150)
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    print("(" + str(20-i)  + ", " + str(i) + ") " + str(accuracy_score(y_test, y_pred)))
```

Neuron combination	Test accuracy
(20)	0.7456647398843931
(19, 1)	0.6859344894026975
(18, 2)	0.720616570327553
(17, 3)	0.7032755298651252
(16, 4)	0.6859344894026975
(15, 5)	0.7456647398843931
(14, 6)	0.74373795761079
(13, 7)	0.7957610789980732
(12, 8)	0.7225433526011561
(11, 9)	0.7649325626204239
(10, 10)	0.6936416184971098
(9, 11)	0.7148362235067437
(8, 12)	0.7148362235067437
(7, 13)	0.6917148362235067
(6, 14)	0.7418111753371869
(5, 15)	0.6878612716763006
(4, 16)	0.7244701348747592
(3, 17)	0.7244701348747592
(2, 18)	0.6936416184971098
(1, 19)	0.6859344894026975

The highest accuracy score was 0.7957610789980732, the combination was relatively balanced with (13, 7). The lowest accuracy score was 0.6859344894026975 there were three combination with this score, they were (19, 1), (16, 4) and (1, 19). These are relatively imbalanced hidden layers. The median is 0.7177264 and the variance is relatively low at 0.0009047826. The IQR is 0.04913295.

1) Give reasons for any difference in trends between the tables you produced in parts 2 and 4 of this question. **(10 marks)**

Although the datasets were vastly different there were several similarities between the results shown in part 2 and 4. We could see that imbalance of neurons between hidden layers definitely affected the results. The accuracy scores were also similar with the pima Indians diabetes results having a median accuracy score of 0.7229437, compared to the car evaluation median accuracy score of 0.7177264. Although there were similarities there were differences in the trends of the data. The variance for pima Indians diabetes data is 0.001403104 which is more than the car evaluations of 0.0009047826. This indicates the accuracy scores for car evaluation are generally more consistent than the pima Indians diabetes dataset. However, the IQR for the pima Indians dataset was 0.02597403 compared to 0.04913295. This indicates the Car evaluation has data that is more distant from the middle values, indicating that it may have more outliers. There could be multiple reasons for these differences. One possible reason is the difference in the data type. The pima diabetes dataset contains only numerical data, while the car evaluation dataset contains categorical text data. These had to be changed to discrete values, which may have changed the context of the data, making the data more susceptible to changes in accuracy scores. Also, a common issue with categorical data that affects results is the class imbalance. Below is a table of the class distribution for car evaluation dataset.

class	instances	%
unacc	1210	70.023%
acc	384	22.222
good	69	3.993
v-good	65	3.732

As we can see unacc has the most instances with 70.023% of the instance. This causes an imbalance. Which may cause the model to become unstable, which would explain the higher IQR and outliers. A solution could be to group smaller categories together, such as group good and v-good together. There is other solution such as under-sampling or getting more data. There can be other issues with categorical data as well such as such as issues with the partition algorithm which could lead to unstable models and therefore more outliers. I think the fact that the car evaluation dataset used categorical data, explains adequately the reasons for the higher IQR and outliers, while still having a higher variance.