ANLY 530 Machine Learning I, Laboratory #2

Introduction

As you learned in ANLY 500 analytics can be categorized as descriptive, predictive or prescriptive. Machine learning is engaged in all categories of analytics. We've obviously done descriptive and predictive analytics. Prescriptive analytics is what happens when you take predictions made and use them to make strategic changes, e.g. to a business model in order to refocus or enhance the model. Part of this process is also handling risk. Analytics is frequently used to consider a variety of types of risk; credit, sales, fraud related, etc. One of the challenges to prescriptive analytics is the complexity in integrating multiple streams of data, e.g. business, risk, environmental, contextual, stakeholder, and so much more.

This laboratory will get you started thinking about pieces of what is required for prescriptive analytics and in particular how machine learning can be used to support that function. This laboratory will specifically look at credit risk. We'll define credit risk as the risk of default on a debt due to a borrower failing to make the required payments in a timely manner. In this definition, the lender assumes the risk of losing both the principal and the anticipated interest on that principal. For the problems in this laboratory we will use a bank as the financial institution. The bank will analyze customer data to predict which customers might be credit risks. These predictions will then feed into risk management.

This particular laboratory focuses on the use of Bayesian methods, specifically Naïve Bayesian Classifiers. You won't necessarily have to read the uploaded papers or the websites listed below to complete this laboratory. They are available only if you want more information. These two papers have been uploaded to Moodle and are intended to provide you with additional background and information about the different challenges and algorithms used for credit scoring; "Guide to Credit Scoring in R' by Sharma, and "Classification methods applied to credit scoring: A systematic review and overall comparison" by Louzada. To specifically learn more about Bayesian network learning and the bnlearn package we'll use for this laboratory there are many websites including:

- http://www.bnlearn.com/
- https://cran.r-project.org/web/packages/bnlearn/index.html
- https://www.r-project.org/nosvn/conferences/useR-2013/Tutorials/Scutari.html

A data set has been provided for you, creditData.txt. We've used many data sets already. You'll need to **start by loading the data into R and looking at the data set**. Since we've done this many times I won't include a separate step for reading in the data and examining it in this laboratory. That is, the equivalent of "Step 1" in Laboratory #1 and is not explicitly included in this laboratory. A description of the data set is included at the end of this laboratory.

One of the differences between data mining and machine learning is that machine learning focuses more on improving performance, whether that is improving the performance of an algorithm or increasing learning, than it (machine learning) does on the specific results obtained from running an algorithm (as does data mining). In this laboratory we'll begin to consider that and take a "brute force" approach to improving performance and/or increasing learning. We'll begin by using R functions for a Naïve Bayes Classifier as black-box tools and then we'll apply common techniques for improvement. I'll call the original, unimproved application of the algorithms Part 1. The brute force part is that we'll apply techniques to improve performance individually and manually to the original algorithms. I'll call this Part 2.

Laboratory 2: Naïve Bayes Classifiers, Part 1

Step 1: Exploring and Preparing the Data Once you have read in the data set you'll need to use the naive_bayes() function to establish a Naïve Bayes Classifier for the data. For many functions to work correctly there cannot be any missing values in the data set. So far we haven't used any data sets that require a lot of processing or looking to determine if there are missing values. For this data set it is more difficult to just visually ensure that there are no missing values. So, you could use the following command built on the *sum()* and *is.na()* functions. It will return a value that is the sum (total number of) all missing values:

```
sum(is.na(creditData))
```

Q1- (one thing to make sure you do!): Remember that the class variable needs to be a categorical data type in order to build a Naïve Bayes Classifier. This means that you'll need to convert your class variable.

Next, use a 75%/25% split for training and test data, i.e. use 75% of the records for the training set and 25% of the records for the test set. Report the number of missing values you find in the data in your results report. Use the randomization seed of 12345.

Q2- Compute the percentage of both classes similar to what you did in lab 1 and see if the distribution of both classes preserved for both training and testing data.

Step 2: Training a Model on the Data Use the naive_bayes() function on your training data to build the Naïve Bayes Classifier. In your results report include what percentage of the training records your Naïve Bayes Classifier reported as credit worthy.

Step 3: Evaluating Model Performance Now, use the processes we have completed before to evaluate your Naïve Bayes Classifier. In your results report include your confusion table showing your results. Be sure to include 2-3 sentences explaining what those results actually mean.

Laboratory 2: Naïve Bayes Classifiers, Part 2

In this part of the laboratory we'll manually work to improve the performance of the Naïve Bayes classifier. There are several ways of improving the performance of an algorithm, e.g. pre-processing the data can make a very big difference in the performance of any algorithm. One category of methods to improve performance of algorithms is referred to as feature selection. Two types of feature selection and more are reported in the article on the R-Bloggers webpage: https://www.r-bloggers.com/introduction-to-feature-selection-for-bioinformaticians-using-r-correlation-matrix-filters-pca-backward-selection/. I will follow the approach using correlations on that webpage to determine if there are variables in the data set that if they are eliminated would improve the performance of the Naïve Bayes Classifier. You can choose whatever method you want to use to select the features (variables) to keep. Be sure to include a paragraph or so in your results report describing what/how you tried to increase the Naïve Bayes algorithm's performance; and, include any gains in performance in that report too.

Step 1: Exploring and Preparing the Data The first approach I'll introduce is using the correlation matrix to perform feature, i.e. variable selection. As always, there are several R packages with functions for computing a correlation matrix. (I could suggest using cor() function to compute the correlation matrix and to visualize the correlation matrix the corrplot() function in the Corrplot package. Hint: if you want to use the corrplot() function you will probably need to install and attach the Corrplot package.) But, before doing anything you need to determine if any pre-processing of the data is required, e.g. scaling. Be sure to report any pre-processing of the data in your results report. If you do use corrplot() be sure to include your plot as a figure in your results report.

Next, you'll need to pull out those features, or variables, with correlations higher than a specified threshold. What is the best threshold to use? It depends on whether you're most interested in achieving the highest level of classification with the fewest errors (increasing performance) or in balancing this with the variables that reduce overall risk the most. That is, if you eliminate too many features (variables) you may end up with a classifier that misses some critical piece of evidence that actually increases risk even though the classifier performs more perfectly.

There is a function, *findCorrelation()*, that is part of the caret package. This automates the process of determining which features of a data set are correlated above/below a cutoff value. It is very helpful to be able to use this function to create a feature set of interest. In fact, I believe caret may be an extremely useful R package with many great functions. I'm uploading a copy of the cran.r-project pdf file on caret to Moodle.

However, I found that this package is also one of the most difficult to install I've found yet. When you install packages automatically from the pull-down menu in RStudio you can check the box to include "Dependencies". Apparently this is not sufficient to install caret. You must use the command line in the console and also include "Suggests". Do this as:

```
> install.packages("caret", dependencies = c("Depends", "Suggests"))
```

But even more is involved. This still does not install a few packages that caret requires. I found I also had to install: minqa; nloptr; and, MatrixModels. Use the same command and install each of these including "Suggests". You might also have to independently install the packages: colorspace; and, ggplot2. If everything you need is installed you will not get any error messages when you execute the command:

```
> library(caret)
```

Once caret is installed and attached properly, you can use the findCorrelation() function as follows:

```
> highlycor <- findCorrelation(m, 0.30)
```

where 0.30 is the cutoff value I tried as a first trial. You'll need to determine the cutoff value you believe is most appropriate and include that in your results report. This is just one of invoking feature selection. As I said before you can use whatever method you want to try to improve the performance of the original Naïve Bayes Classifier. The following steps are just hints and suggestions based on my results.

There is an "order of operations" required to get all this done correctly. That is:

1. Randomize the data

   ```
   > credit_rand <- creditData[order(runif(1000)), ]
   ```

2. Scale the data

   ```
   > creditDataScaled <- scale(credit_rand[,2:ncol(credit_rand)], center=TRUE, scale = TRUE)
   ```

3. Compute the correlation matrix (note that this does not include the class variable)

   ```
   > m <- cor(creditDataScaled)
   ```

4. Determine the threshold to use for feature (variable) selection (I'll continue to use 0.30 as an example.) and perform the feature selection.

   ```
   > highlycor <- findCorrelation(m, 0.30)
   ```

5. Recombine the class variable with the highly correlated credit data and split into training and test data sets – BUT before you do this you will need to go back to the unscaled data to get a feature data set because the naive_bayes() function seems to do strange things with scaled data when the parameter centered equals TRUE!

```
> filteredData <- credit_rand[, -(highlycor[5]+1)]
> filteredTraining <- filteredData[1:750, ]
> filteredTest <- filteredData[751:1000, ]
```

plus, do all the other sorts of things like initializing the overall process for reproducibility etc.

Step 2: Training a Model on the Data At this point we follow the same process that we have before.

6. Build and evaluate the Naïve Bayes Classifier as usual

```
> nb_model <- naive_bayes(Creditability ~ ., data=filteredTraining)
```

Step 3: Evaluating Model Performance

7. Evaluate the Naïve Bayes Classifier as usual

```
> filteredTestPred <- predict(nb_model, newdata = filteredTest)
> table(filteredTestPred, filteredTest$Creditability)
```

Q3- What is the accuracy this time? Be sure to include in your results report whether or not, after all this work, the performance of your Naïve Bayes Classifier was improved.

Laboratory 2: Support Vector Machine, Part 3

Step 1: Collecting the Data

We'll look at support vector machines from the perspective of optical character recognition (OCR). We'll use the letterdata.csv file also from the UCI Data Repository. We can simply read it in and look at the structure as follows:

```
> letters <- read.csv("letterdata.csv")
> str(letters)
'data.frame':      20000 obs. of  17 variables:
 $ letter: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10 13 ...
 $ xbox  : int  2 5 4 7 2 4 4 1 2 11 ...
 $ ybox  : int  8 12 11 11 1 11 2 1 2 15 ...
 $ width : int  3 3 6 6 3 5 5 3 4 13 ...
 $ height: int  5 7 8 6 1 8 4 2 4 9 ...
 $ onpix : int  1 2 6 3 1 3 4 1 2 7 ...
 $ xbar  : int  8 10 10 5 8 8 8 8 10 13 ...
 $ ybar  : int  13 5 6 9 6 8 7 2 6 2 ...
 $ x2bar : int  0 5 2 4 6 6 6 2 2 6 ...
 $ y2bar : int  6 4 6 6 6 9 6 2 6 2 ...
 $ xybar : int  6 13 10 4 6 5 7 8 12 12 ...
 $ x2ybar: int  10 3 3 4 5 6 6 2 4 1 ...
 $ xy2bar: int  8 9 7 10 9 6 6 8 8 9 ...
 $ xedge : int  0 2 3 6 1 0 2 1 1 8 ...
 $ xedgey: int  8 8 7 10 7 8 8 6 6 1 ...
 $ yedge : int  0 4 3 2 5 9 7 2 1 1 ...
 $ yedgex: int  8 10 9 8 10 7 10 7 7 8 ...
```

Step 2: Preparing the Data

The only thing we need to do to prepare this data set is to subset it into our training and test data sets. There are 20,000 observations so we have considerable flexibility in deciding how many to put in our training set and how many to keep for testing. In class we decided to use 18,000. We'll stick with that number now.

```
> letters_train <- letters[1:18000, ]
> letters_test <- letters[18001:20000, ]
```

Step 3: Training a Model on the Data

Again, we'll need to install and attach a package to complete this analysis, i.e. the kernlab package. The actual command is the ksvm() command in this package. It requires us to set up a syntax like the formulas we've used before fn(y ~ x, …). We'll also use the "vanilladot" kernel for this analysis. For more information on the ksvm() command and what the "vanilladot" kernel is use the help("ksvm") command in RStudio. It may take some time for the algorithm to run. You'll see a red "stop sign" icon at the top of the console frame in RStudio while it is running. The complete command is:

```
> letter_classifier <- ksvm(letter ~ ., data = letters_train, kernel = "vanilladot")
```

As an initial indication of how the algorithm performed simply enter the object name you used for the output from the ksvm() command at the command prompt, i.e.:

```
> letter_classifier
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1
```

```
Linear (vanilla) kernel function.

Number of Support Vectors : 7886

Objective Function Value : -15.3458 -21.3403 -25.7672 -6.8685 -8.8812 -35.9555 …
… -151.3409 -90.0329 -27.9491 -42.4688 -12.5118 -26.4828 -2.0045 -62.195 -9.1662 -178.4616 -1.9406 -1.9871 -1
1.3982 -0.5214 -29.6136 -35.0449 -6.7569
Training error : 0.1335
```

That is, we have an initial training error of a bit over 13%.

Step 4: Evaluating Model Performance

Again, we’ll develop a confusion table to determine how well our model performed overall. Since we are looking at letters of the alphabet we’ll have a considerable number of columns and rows to our table. You might want to adjust the size of the console frame accordingly. The commands to run the evaluation and output the table are:

```
> letter_predictions <- predict(letter_classifier, letters_test)
> table(letter_predictions, letters_test$letter)

letter_predictions  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O  P  Q  R  S  T  U  V  W  X  Y  Z
                 A 73  0  0  0  0  0  0  0  0  1  0  0  0  0  3  0  4  0  0  1  2  0  1  0  0  0
                 B  0 61  0  3  2  0  1  1  0  0  1  1  0  0  0  2  0  1  3  0  0  0  0  0  0  0
                 C  0  0 64  0  2  0  4  2  1  0  1  2  0  0  1  0  0  0  0  0  0  0  0  0  0  0
                 D  2  1  0 67  0  0  1  3  3  2  1  2  0  3  4  2  1  2  0  0  0  0  0  0  1  0
                 E  0  0  1  0 64  1  1  0  0  0  2  2  0  0  0  0  2  0  6  0  0  0  0  1  0  0
                 F  0  0  0  0  0 70  1  1  4  0  0  0  0  0  0  5  1  0  2  0  0  1  0  0  2  0
                 G  1  1  2  1  3  2 68  1  0  0  0  1  0  0  0  0  4  1  3  2  0  0  0  0  0  0
                 H  0  0  0  1  0  1  0 46  0  2  3  1  1  1  9  0  0  5  0  3  0  2  0  0  1  0
                 I  0  0  0  0  0  0  0  0 65  3  0  0  0  0  0  0  0  0  2  0  0  0  0  2  1  0
                 J  0  1  0  0  0  1  0  0  3 61  0  0  0  0  1  0  0  0  1  0  0  0  0  1  0  4
                 K  0  1  4  0  0  0  0  5  0  0 56  0  0  2  0  0  0  4  0  1  2  0  0  4  0  0
                 L  0  0  0  0  1  0  0  1  0  0  0 63  0  0  0  0  0  0  0  0  0  0  0  0  0  0
                 M  0  0  1  0  0  0  1  0  0  0  0  0 70  2  0  0  0  0  0  0  1  0  6  0  0  0
                 N  0  0  0  0  0  0  0  0  0  0  0  0  0 77  0  0  0  1  0  0  1  0  2  0  0  0
                 O  0  0  1  1  0  0  0  1  0  1  0  0  0  0 49  1  2  0  0  0  1  0  0  0  0  0
                 P  0  0  0  0  0  3  0  0  0  0  0  0  0  0  2 69  0  0  0  0  0  0  0  0  1  0
                 Q  0  0  0  0  0  0  3  1  0  0  0  2  0  0  2  1 52  0  1  0  0  0  0  0  0  0
                 R  0  4  0  0  1  0  0  3  0  0  3  0  0  0  1  0  0 64  0  1  0  1  0  0  0  0
                 S  0  1  0  0  1  1  1  0  1  1  0  0  0  0  0  0  6  0 47  1  0  0  0  1  0  6
                 T  0  0  0  0  1  1  0  0  0  0  1  0  0  0  0  0  0  0  1 83  1  0  0  0  2  2
                 U  0  0  2  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0 83  0  0  0  0  0
                 V  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0  0  0 64  1  0  1  0
                 W  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  3 59  0  0  0
                 X  0  1  0  0  1  0  0  1  0  0  1  4  0  0  0  0  0  1  0  0  0  0  0 76  1  0
                 Y  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0  0  0  1  0  0  0  1 58  0
                 Z  1  0  0  0  2  0  0  0  0  2  0  0  0  0  0  0  0  0  5  1  0  0  0  0  0 70
```

As seen in the table, most of the predictions matched the actual letters. We can get the evaluation data more succinctly by the following:

```
> agreement <- letter_predictions == letters_test$letter
> table(agreement)
agreement
FALSE  TRUE
  321  1679
```

This tells us that the classification was correct in 1,679 out of our 2000 test records. As before, we'll defer improving model performance until later in the course.

Now compute the accuracy which gave 83.95 to me.

Q4- We may be able to do better than this by changing the Kernels. Try Polynomial and RBF kernels to improve the result.

Laboratory 2: News popularity, Part 4

Now apply the Naïve Bayes classifier and SVM that you saw in Parts 1 through 3 on News popularity data set from lab 1.

Q5- Do you see any improvement compared to last three techniques? Please completely explain your results and analysis.

Summary of covariates for the dataset "Determining the solidness of borrowers via creditscoring"

Notice:
The given score for the categorical covariates is based on the assessment of experienced bank specialists dealing with credits and is contained in the downloadable file. The description of the variables is consistent with "Fahrmeir / Hamerle / Tutz (1996, 2nd ed.): *Multivariate statistische Verfahren.* de Gruyter, Berlin. p. 390 ff."

Variable	**Description**	**Categories**	**Score**	**rel. frequency in % for**	
				good credits	**bad credits**
Creditability	Creditability: 1 : credit-worthy 0 : not credit-worthy				
Account Balance	Balance of current account	no balance or debit	2	35.00	23.43
		0 <= ... < 200 DM	3	4.67	7.00
		... >= 200 DM or checking account for at least 1 year	4	15.33	49.71
		no running account	1	45.00	19.86
Duration of Credit (month)	Duration in months (metric)				
Duration of Credit (month)	Duration in months (categorized)	<=6	10	3.00	10.43
		6 < ... <= 12	9	22.33	30.00
		12 < ... <= 18	8	18.67	18.71
		18 < ... <= 24	7	22.00	22.57
		24 < ... <= 30	6	6.33	5.43
		30 < ... <= 36	5	12.67	6.86
		36 < ... <= 42	4	1.67	1.71
		42 < ... <= 48	3	10.67	3.14
		48 < ... <= 54	2	0.33	0.14
		> 54	1	2.33	1.00
Payment Status of Previous	Payment of previous credits	no previous credits / paid back all previous	2	56.33	51.57

Credit		credits			
		paid back previous credits at this bank	4	16.67	34.71
		no problems with current credits at this bank	3	9.33	8.57
		hesitant payment of previous credits	0	8.33	2.14
		problematic running account / there are further credits running but at other banks	1	9.33	3.00
Purpose	Purpose of credit	new car	1	5.67	12.29
		used car	2	19.33	17.57
		items of furniture	3	20.67	31.14
		radio / television	4	1.33	1.14
		household appliances	5	2.67	2.00
		repair	6	7.33	4.00
		education	7	0.00	0.00
		vacation	8	0.33	1.14
		retraining	9	11.33	9.00
		business	10	1.67	1.00
		other	0	29.67	20.71
Credit Amount	Amount of credit in "Deutsche Mark" (metric)				
Credit Amount	Amount of credit in DM (categorized)	<=500	10	1.00	2.14
		500 < ... <= 1000	9	11.33	9.14
		1000 < ... <= 1500	8	17.00	19.86
		1500 < ... <= 2500	7	19.67	24.57
		2500 < ... <= 5000	6	25.00	28.57
		5000 < ... <= 7500	5	11.33	9.71
		7500 < ... <= 10000	4	6.67	3.71
		10000 < ... <= 15000	3	7.00	2.00
		15000 < ... <= 20000	2	1.00	0.29
		> 20000	1	0.00	0.00

Value Savings/Stocks	Value of savings or stocks	< 100,- DM	2	11.33	9.86
		100,- <= ... < 500,- DM	3	3.67	7.43
		500,- <= ... < 1000,- DM	4	2.00	6.00
		>= 1000,- DM	5	10.67	21.57
		not available / no savings	1	72.33	55.14
Length of current employment	Has been employed by current employer for	unemployed	1	7.67	5.57
		<= 1 year	2	23.33	14.57
		1 <= ... < 4 years	3	34.67	33.57
		4 <= ... < 7 years	4	13.00	19.29
		>= 7 years	5	21.33	27.00
Instalment per cent	Installment in % of available income	>= 35	1	11.33	14.57
		25 <= ... < 35	2	20.67	24.14
		20 <= ... < 25	3	15.00	16.00
		< 20	4	53.00	45.29
Sex & Marital Status	Marital Status / Sex	male: divorced / living apart	1	6.67	4.29
		female: divorced / living apart / married	2	11.33	10.29
		male: single	2	25.00	18.43
		male: married / widowed	3	48.67	57.43
		female: single	4	8.33	9.57
Guarantors	Further debtors / Guarantors	none	1	90.67	90.71
		Co-Applicant	2	6.00	3.29
		Guarantor	3	3.33	6.00
Duration in Current address	Living in current household for	< 1 year	1	12.00	13.43
		1 <= ... < 4 years	2	32.33	30.14
		4 <= ... < 7 years	3	14.33	15.14
		>= 7 years	4	41.33	41.29
Most valuable available asset	Most valuable available assets	Ownership of house or land	4	22.33	12.43

		Savings contract with a building society / Life insurance	3	34.00	32.86
		Car / Other	2	23.67	23.00
		not available / no assets	1	20.00	31.71
Age (years)	Age in years (metric)				
Age (years)	Age in years (categorized)	0 <= ... <= 25	1	26.67	15.71
		26 <= ... <= 39	2	47.33	52.72
		40 <= ... <= 59	3	21.67	26.14
		60 <= ... <= 64	5	2.33	3.00
		>= 65	4	2.00	2.43
Concurrent Credits	Further running credits	at other banks	1	19.00	11.71
		at department store or mail order house	2	6.33	4.00
		no further running credits	3	74.67	84.29
Type of apartment	Type of apartment	rented flat	2	62.00	75.43
		owner-occupied flat	3	14.67	9.14
		free apartment	1	23.33	15.57
No of Credits at this Bank	Number of previous credits at this bank (including the running one)	one	1	66.67	61.86
		two or three	2	30.67	34.43
		four or five	3	2.00	3.14
		six or more	4	0.67	0.57
Occupation	Occupation	unemployed / unskilled with no permanent residence	1	2.33	2.14
		unskilled with permanent residence	2	18.67	20.57
		skilled worker / skilled employee / minor civil servant	3	62.00	63.43
		executive / self-employed / higher civil servant	4	17.00	13.86
No of dependents	Number of persons	0 to 2	2	84.67	84.43

	entitled to maintenance	3 and more	1	15.33	15.57
Telephone	Telephone	no	1	62.33	58.43
		yes	2	37.67	41.57
Foreign Worker	Foreign worker	yes	1	1.33	4.71
		no	2	98.67	95.29

The covariates M1, M2, M5, M7, M8 are consistent with the description in "Fahrmeir / Hamerle / Tutz (1985, 1st ed.): *Multivariate statistische Verfahren.* de Gruyter, Berlin. p. 285 ff."