Josep Puig Ruiz

Final Project Report

EN.553.688: Financial Computing, Fall 2019

Instructor: Dr. Daniel Q. Naiman

Contents of the report:

1. Introduction

The goal of this project is to create prediction models related to loan duration and loan revenues based on real data. The data has been split in 3 groups: dataset 0, dataset1, and dataset 2. For the two first datasets there is performance data. The objective is to use these two datasets (0 and 1) to find which models would better predict the loan duration and revenues for dataset 2.

This report aims to explain the ideas behind this project, the methodology followed, and the results that can be derived. The explanations in this report, together with the comments on the Jupyter Notebook, should provide the reader with enough understanding of the author's ideas so that she can execute, modify, or improve the provided code.

2. Preprocessing: assumptions and methodology

The data is read using the *read_csv* method, from the CSV files, and store it in several dataframes:

dfA0	dfCF0	dfA1	dfCF1	dfA2
Acquisition data, dataset 0	Performance data, dataset 0	Acquisition data, dataset 1	Performance data, dataset 1	Acquisition data, dataset 2

The index is set to be the loan ID ("*LID*") so that I make sure that I can identify at all times each loan from both acquisition and performance data.

There are two functions which allow to preprocess these dataframes:

- *preprocessdfA()*, to preprocess the dataframes containing information about acquisition (i.e., dfA0, dfA1 and dfA2)
- *preprocessdfCF()*, to preprocess the dataframes containing information about performance (i.e., dfCF0 and dfCF1)

a) Preprocessing acquisition data (*preprocessdfA()*)

To deal with date, an arbitrary date has been chosen ("2000-01-01"), and the *ORIG_DATE* feature has been converted into "days since 2000-01-01" using *datetime.strptime()*. The format of the data is "%Y-%d-%m". This conversion can be seen in the following table:

Original value of *ORIG_DATE*	New value of *ORIG_DATE*
…	…
1999-30-12	-2
1999-31-12	-1
2000-01-01	0
2000-02-01	1
2000-03-01	2
…	…

There are several features of the loans which have decided to not use for the predictions: *FIRST_PMT_DATE* (I am already considering time in *ORIG_DATE*), *ZIP*, *MORT_INS_PCT*,

MORT_INS_TYPE, SELLER, STATE, OCC_STATUS, OCHANNEL, CO_CREDIT_SCORE, and *PROD_TYPE*. This has been done because including these features have not really improved the accuracy of the predictions, whilst they did increase the number of variables and thus the execution time.

A few of the loans were including some missing values in features. Originally this was solved by dropping those loans with missing data (which accounted for a small percentage of the total data). However, this approach was later changed to simply filling the missing values with the mean of the values for that feature from the rest of the loans. The *df.fillna()* method was used to achieve this. For instance, for the number of borrowers, the following code was used. This has been done for *NBORROWERS, CLTV, LTV, DTI,* and *CREDIT_SCORE*.

```
dfA['NBORROWERS'].fillna((dfA['NBORROWERS'].mean()), inplace=True)
```

For the categorical features, in order to use them with the different algorithms, the *LabelEncoder* class has been used, to convert into numerical features. This assigns values *0, 1, …, n-1* for each categorical feature, assuming there are *n* different possible values. This may introduce some bias in the models by assigning closer values to categorical features that have no relation. In order to prevent this, *OneHotEncoder* was tested. This encoding system creates new dummy variables for each of the possible values and assigns 1 to these loans whose value is that one, and 0 otherwise. This helps with the abovementioned bias but introduces a lot of complexity in the understanding of the underlying variables and increases the execution time whilst not really improving the results, so after some testing it has been discarded in favor of *LabelEncoder*.

b) Preprocessing performance data (*preprocessdfCF()*)

Preprocessing the performance data has consisted in two parts:

First, the goal is to obtain, for each loan, and for each $m = 12, 24, 36, 48,$ and 60 months, a binary variable $p_{LID}(m)$ which is 1 if the specific loan is still active m months after acquisition time. Therefore, 6 values need to be calculated for each loan.

Second, the goal is to obtain, for each loan, and for each $m = 12, 24, 36, 48,$ and 60 months, a continuous variable $r_{LID}(m)$ which calculates the cumulative revenue up to and including month m after acquisition time. Thus, another 6 values need to be calculated for each loan.

In order to determine the value of these 12 variables for each lone, the *preprocessdfCF()* function has been created. It pretty much creates a pivot table out of the cash flow data:

```
Y = pd.pivot_table(dfCF, index = ['LID'],
                         aggfunc={'LOAN_AGE':[p12, p24, p36, p48, p60],
                                  'PAYMENT':[r12, r24, r36, r48, r60]})
```

This function uses loan ID (*LID*) as index, and uses the custom functions p12, p24, …, p60, r12, r24, …, r60 which return the desired values.

For instance, p12 looks like:

```
def p12(a):
    return 1 if len(a)>=12 else 0
```

For a given array a of cash flows *a* given by the pivot_table (i..e., for a specific *LID*), it simply returns 1 if the length of the cash flows array is greater or equal than 12, and 0 otherwise. Similar for the rest of maturities *m*.

And r12 looks like:

```
def r12(a):
    return sum([a.iloc[i] for i in range(min(13, len(a)))])
```

For a given array a of cash flows *a* given by the pivot_table (i..e., for a specific *LID*), if it has more than 12 values, then it returns the summation of the first 12 values, and if it has less than 12 values, then it returns the summation of all values. Therefore, it is calculating the cumulative revenue up to month 12. Similar for the rest of maturities *m*.

Hence, *preprocessdfCF()* simply calculates the required values from the cashflows so that several machine learning algorithms can try to predict them.

The processed dataframes are stored in X0 and y0 (for acquisition and performance data, respectively, from dataset 0), and in X1 and y1 (similarly, but for from dataset 1).

3. Estimating loan duration from datasets 0 and 1

a. Assessment of accuracy

The first prediction is about the chance of a loan being active for at least m months, for $m = 12, 24, 36, 48, \text{and } 60$. In order to find the better method for each of them, the following methodology has been followed. For each one of datasets 0 and 1, several algorithms – which will be explained in next section – have been tested by randomly splitting the data into train (80%) and test (20%), training the algorithm using the train data, and assessing the accuracy of the methods on the test data.

In order to assess the accuracy of each algorithm, a score function has been defined. Let $\hat{p}_i$ be the predicted value by the algorithm for each of the loan ID i, for $i = 0, \dots, N-1$, and let Y_i be the real value of the variable for each loan ID, for the same loan ID i. Then, the score is the average of the absolute difference between the predicted value and the actual value:

$$Score = \frac{\sum_{i=0}^{N-1} |\hat{p}_i - Y_i|}{N}$$

It is easy to see that, with this definition of score, the lower the score is, the better the algorithm is. The goal is thus **to find the algorithm with the lowest score.** This will be the algorithm which will be used to predict p(m) for the dataset 2.

On a double for loop (the first one on using dataset 0 or 1, and the second one on using 12, 24, …, 60 months), each algorithm is trained and tested. The results will be shown in c). For each value,

a function called *getScores(i, pX)* is called and inserted into an output dataframe which stores the scores:

```
pXlist = ["p60", "p48", "p36", "p24", "p12"]

for pX in pXlist:
    for i in [1, 0]:
        nameColumn = "score"+str(i)+pX
        dfP.insert(1, nameColumn, getScores(i, pX))
```

The function *getScores(i, pX)* find the score of using each algorithm to predict values.

b. Algorithms tested

- Logistic Regression
  - Classifying as 0 or 1
  - Providing a probability from 0 to 1
- Random Forest
- Decision Tree
- Gaussian Naïve Bayes
- Support Vector Machine[1]

As a brief comment, logistic regression can provide both a probability (likelihood) between 0 and 1 of being active or not, or simply a value of 0 (inactive) or 1 (active). The first one can be converted into the second one by providing a probability threshold. These two approaches have been kept differentiated due to a simple reason. For the performance criterion for this project, it will be calculated as $|\hat{p} - Y|$. It is possible that using this criterion the performance would be considered better by just assigning a label (0 or 1) rather than using a continuous probability. Hence, if it comes to that, I might provide the label 0 or 1 as the requested chance p(m) instead of just the probability.

The performance of the algorithms is shown in **section 5**.

4. Estimating loan revenues from datasets 0 and 1

a. Assessment of accuracy

The second prediction is about the cumulative revenue for a loan at loan age *m* months. A similar methodology has been followed. For each one of datasets 0 and 1, several variations of algorithms – which will be explained in next section – have been tested by randomly splitting the data into train (80%) and test (20%), training the algorithm using the train data, and assessing the accuracy of the methods on the test data.

[1] SVM has been initially implemented but the execution time is substantially greater than for the rest of algorithms whilst providing similar results. What's more, the results from SVM are also irregular: behaving quite nicely for some cases but ridiculously wrong for other cases. Hence, it has been ruled out after testing.

In order to assess the accuracy of each algorithm, an error function has been defined. Let $\hat{r}_i$ be the predicted cumulative revenue by the algorithm for each of the loan ID i, for $i = 0, \dots, N-1$, and let Y_i be the real cumulative revenue for each loan ID, for the same loan ID i. Then, the error is the average of the absolute difference between the predicted value and the actual value:

$$Error = \frac{\sum_{i=0}^{N-1} |\hat{p}_i - Y_i|}{N}$$

Hence, the goal is **to find the algorithm with the lowest error.** This will be the algorithm which will be used to predict r(m) for the dataset 2.

On a double for loop (the first one on using dataset 0 or 1, and the second one on using 12, 24, …, 60 months), each algorithm is trained and tested, and a score is found. For each value, a function called *getScores(i, rX)* is called and inserted into an output dataframe which stores the scores:

```
rXlist = ["r60", "r48", "r36", "r24", "r12"]
for rX in rXlist:
    for i in [1, 0]:
        nameColumn = "error"+str(i)+rX
        dfR.insert(1, nameColumn, getScores(i, rX))
```

The function *getScores(i, rX)* find the score of using each algorithm to predict values.

b. Algorithms tested

- Linear Regression, with initial variables
- Linear Regression, with Polynomial Features = 2
- Linear Regression, with Polynomial Features = 2 but only interactions
- Linear Regression, with Polynomial Features = 3
- Linear Regression, with Polynomial Features = 4

Basically, only linear regression is used, but with different variations, mainly using Polynomial Features of different degrees, and considering or not only interactions. The goal was to see if there was a combination of initial variables which proved to be a really good indicator (predictor) of r(m).

Warning: As the degree of polynomial features is increased, the number of variables to be considered dramatically increases. This makes the execution time of the case with degree 3 of roughly 5 minutes, and for degree 4 of roughly 50 minutes!

The performance results of the algorithms are shown in the next section (**section 5**).

5. Performance of the algorithms

- Algorithms to predict p(m)

Let $\hat{p}_i$ be the predicted value by the algorithm for each of the loan ID i, for $i = 0, \dots, N-1$, and let Y_i be the real value of the variable for each loan ID, for the same loan ID i. Then, the score is the average of the absolute difference between the predicted value and the actual value. Note that lower scores are better.

$$Score = \frac{\sum_{i=0}^{N-1} |\hat{p}_i - Y_i|}{N}$$

The score for each of the algorithms tested can be found on the next table. The notation is "*score X p m*" where X can be 0 (dataset 0) or 1 (dataset 1), and m can be 12, 24, …, 60 months.

	ML Algorithm	score0p12	score1p12	score0p24	score1p24	score0p36	score1p36	score0p48	score1p48	score0p60	score1p60
0	Logistic Regression	0.145754	0.146201	0.392712	0.397535	0.218472	0.219845	0.141028	0.140676	0.101491	0.101779
1	Logistic Regression with Prob	0.244411	0.245538	0.468857	0.470413	0.315654	0.317952	0.223259	0.224152	0.169683	0.170887
2	Random Forest	0.159455	0.160700	0.378629	0.377224	0.222208	0.223454	0.146457	0.146329	0.105068	0.105930
3	Decision Tree	0.149906	0.151822	0.370868	0.371858	0.226487	0.228180	0.148533	0.147926	0.106186	0.107751
4	Gaussian NB	0.157666	0.161307	0.441574	0.446109	0.230256	0.234982	0.162840	0.164436	0.129595	0.132118

Thus, the chosen algorithms for each $m = 12, 24, \dots, 60$ are the ones which average a lower score between datasets 0 and 1. They can be found on this table:

m (months)	Best performing algorithm for p(m)
12	Logistic Regression
24	Decision Tree
36	Logistic Regression
48	Logistic Regression
60	Logistic Regression

Thus, logistic regression is the algorithm that will be used to estimate $p(m)$ for $m = 12, 36, 48, 60$, whereas decision tree will be the algorithm to predict $p(m)$ for $m = 24$.

- Algorithms to predict r(m)

Let $\hat{r}_i$ be the predicted cumulative revenue by the algorithm for each of the loan ID i, for $i = 0, \dots, N-1$, and let Y_i be the real cumulative revenue for each loan ID, for the same loan ID i. Then, the error is the average of the absolute difference between the predicted value and the actual value:

$$Error = \frac{\sum_{i=0}^{N-1} |\hat{p}_i - Y_i|}{N}$$

The score for each of the algorithms tested can be found on the next table. The notation is "*error X r m*" where X can be 0 (dataset 0) or 1 (dataset 1), and m can be 12, 24, …, 60 months.

	ML Algorithm	error0r12	error1r12	error0r24	error1r24	error0r36	error1r36	error0r48	error1r48	error0r60	error1r60
0	No Polyn Feat	45146.482315	44921.199498	52235.489734	52072.389364	26949.938751	27006.449521	17574.723187	17597.281899	13260.524131	13300.445677
1	Polyn Feat=2	41309.782589	40984.623562	50790.067303	50784.255857	26570.419453	26555.996499	17290.061858	17282.126608	13070.907878	13103.745736
2	Polyn Feat=2, interactions only	41629.786910	41378.536269	51141.600388	51116.554357	26972.102978	27000.679034	17470.170961	17489.102394	13123.272003	13167.677025
3	Polyn Feat=3	40642.218351	40362.870625	50323.844590	50249.051099	26477.067487	26481.922653	17322.874810	17324.209036	13101.987670	13163.846933
4	Polyn Feat=4	40782.356901	40528.019144	50759.435038	50890.275622	26725.777727	26694.735586	17349.238495	17404.920785	13097.156791	13205.929485

Thus, the chosen algorithms for each $m = 12, 24, \dots, 60$ are the ones which average a lower error between datasets 0 and 1. They can be found on this table:

m (months)	Best performing algorithm for r(m)
12	Linear Regression, Polynomial Features = 3
24	Linear Regression, Polynomial Features = 3
36	Linear Regression, Polynomial Features = 3
48	Linear Regression, Polynomial Features = 2
60	Linear Regression, Polynomial Features = 2

Thus, Linear Regression with Polynomial Features = 3 is the algorithm that will be used to estimate $r(m)$ for $m = 12, 24, 36$, whereas Linear Regression with Polynomial Features = 2 will be the algorithm to predict $p(m)$ for $m = 24$. Note that this is an abuse of notation. Polynomial Features = 2 means using polynomial features of degree up to 2

6. Predicting performance data for dataset 2

In order to train the algorithms, the acquisition data for datasets 0 and 1 is concatenated, so that the maximum labeled information is used to train. Likewise, the performance data for datasets 0 and 1 is also concatenated.

With the best performing algorithms abovementioned, the predictions are now performed for dataset 2. The results are stored in a dataframe named *dfOutput*, which has *LID* from dataset 2 as index.

The head of the dataframe *dfOutput* which stores the predictions for the dataset 2 looks like:

LID	p(12)	p(24)	p(36)	p(48)	p(60)	r(12)	r(24)	r(36)	r(48)	r(60)
899993094470	1.0	1.0	0.0	0.0	0.0	19220.285672	19220.285672	19220.285672	34832.315102	34832.315102
899984345565	1.0	0.0	0.0	0.0	0.0	20298.503803	20298.503803	20298.503803	36344.393849	36344.393849
899962467067	1.0	0.0	0.0	0.0	0.0	66227.309269	66227.309269	66227.309269	92823.555975	92823.555975
899956562768	1.0	0.0	0.0	0.0	0.0	41100.489221	41100.489221	41100.489221	89393.455914	89393.455914
899955704832	1.0	1.0	0.0	0.0	0.0	21953.316045	21953.316045	21953.316045	41179.306050	41179.306050

Lastly, the results are exported into the *predictions.csv* file.

Annex: Other approaches

a) Find the threshold which better classifies the predicted values in logistic regression:

```
def classifyLogistic(y_log_prob, threshold):
    y_log = []
    for [a,b] in y_log_prob:
        if a<threshold:
            y_log.append(1)
        else:
            y_log.append(0)
    return y_log

score = []
threshold = [0.01+0.01*i for i in range(99)]
for t in threshold:
    print("Threshold: ", t, "Mean of deviation: ", findScore(y_score, y_log_proba_score, t))
    score.append(findScore(y_score, y_log_proba_score, t))
```

b) Use confusion matrix to assess the prediction's quality:

```
# Decision Tree
treeclass = tree.DecisionTreeClassifier(min_samples_split=25,min_samples_leaf=25)
treeclass = treeclass.fit(X_train, y_train)
y_tree = treeclass.predict(X_test)
CFtree = confusion_matrix(y_test, y_tree)
print("Confusion Matrix for DECISION TREE: \n", CFtree)
print("Accuracy: ", accuracy_CF(CFtree))

# Gaussian NB
gnb = GaussianNB()
gnbclass = gnb.fit(X_train, y_train)
y_gnb = gnbclass.predict(X_test)
CFgnb = confusion_matrix(y_test, y_gnb)
print("Confusion Matrix for Gaussian Naive Bayes: \n", CFgnb)
print("Accuracy: ", accuracy_CF(CFgnb))

# Neural Network
nnwclass = MLPClassifier().fit(X_train, y_train)
y_nnw = nnwclass.predict(X_test)
CFnnw = confusion_matrix(y_test, y_nnw)
print("Confusion Matrix for NEURAL NETWORK: \n", CFnnw)
print("Accuracy: ", accuracy_CF(CFnnw))

# SVM
svmclass = svm.SVC()
svmclass.fit(X_train, y_train)
y_svm = svmclass.predict(X_test)
CFsvm = confusion_matrix(y_test, y_svm)
print("Confusion Matrix for SVM: \n", CFsvm)
```