Capstone Project

Jorge Pinzón

Machine Learning Nanodegree

November 2017

PROBLEM LOAN PREDICTION USING AN ARTIFICIAL NEURAL NETWORK

DEFINITION

Project overview

Banks, and other financial institutions, make their profits by lending money (www.investopedia.com). Every time a loan is issued the institution takes a risk. Risks include the fact that some borrowers, or customers, will not return the money as agreed during the lending process. Lack, or delay, payments results in "Problem loans," or loans where the banks lose money as they cannot recover the original amount, or have to invest additional money during the repaying process (www.iedunote.com). According to the US Department of Treasury, banks deal with problem loans by: renewing or extending the loan terms, extending /adding credit, restructuring the loan, or foreclosure. Regardless of the direction taken, during this process, known as loan workout, the bank is forced to choose the alternative where the recovery is maximized and the risk and expenses are minimized (www.occ.treas.gov). In these circumstances, machine learning techniques and historical data can provide additional information during the initiation of the loan process in order to reduce the risk by identifying customers likely to generate a problem loan.

Problem loan predictions have been tried with several methods in the past. Many datasets in this area are unbalance therefore the solution should include a technique that can this kind of data. Artificial neural networks (ANN) have been found to outperform more traditional machine learning techniques such as support vector machines (SVN) when the data is not balanced (Ren, 2012). The proposed solution in this project includes an ANN classifier that will be trained on the Lending Club dataset (https://www.lendingclub.com/info/download-data.action) from 2017. This solution is optimal for the current data given the unbalance property and the fact that non-neural techniques have been tested with poor results in similar datasets (https://rstudio-pubs-static.s3.amazonaws.com/203258_d20c1a34bc094151a0a1e4f4180c5f6f.html). The ANN solution of this project, is a 3-layer network built on the keras backend tensorflow library on Python 3, two of the most powerful neural network packages available.

Problem Statement

The purpose of this project is to generate a solution to predict whether or not the loan application of a Lending Club customer will be fully paid loan or generate a problem loan for the lender. The proposed solution will be a 3-ANN that will be trained, validated, and tested on the available information for 2017. This approach is optimal given its the unbalance nature of this data and the powerful capabilities of the algorithm.

To build the solution, the data will be obtained from the Lending Club statistics website (https://www.lendingclub.com/info/download-data.action), loaded into Python, cleaned, and submit into the model (Figure 1). The model will be tested against subsets of the original dataset and the results compared to two different models linked to a Kaggle dataset that compiles data from 2007 to 2015 (https://www.kaggle.com/wendykan/lending-club-loan-data).

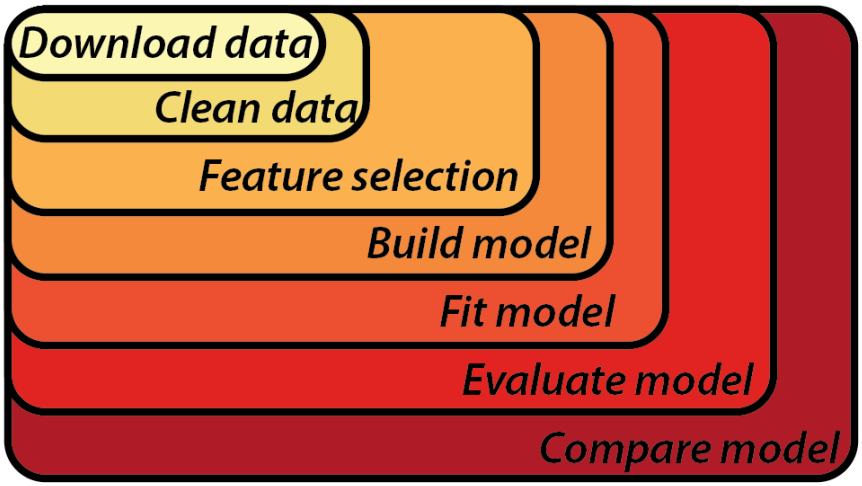

Figure 1 Schematics organization of the procedures taken in the creation of a predictive model for problem loans

Metrics

Originally, the proposal for this project suggested the use of the F-score to evaluate the model and compare to other models on similar datasets. The F-score provides a balance between precision and recall, and therefore complements these two measures, and reduce the accuracy paradox. This metric will be used during the validation and implementation of the model.

As comparisons with benchmark models were executed (See below) both kaggle models failed to identify *problem loans* impeding the calculation of the F-score. To overcome this situation three additional measure were used:

- Accuracy: total No. of correct predictions divided total No. of loans
- Precision: True positives divided True positives + False negative
- Recall: True positive divided True positives + False positives

Using these thee metrics made the comparison between models possible and adding precision and recall reduced the accuracy paradox (https://tryolabs.com/blog/2013/03/25/why-accuracy-alone-bad-measure-classification-tasks-and-what-we-can-do-about-it/), in the absence of F-Score

ANALYSIS

The proposed solution here is divided into seven phases as depicted in figure 1.

Download the data

The data comes from the Lending Club statistics, it includes quarters 1 and 2 of 2017 (https://www.lendingclub.com/info/download-data.action). The data comes in two files, one for each quarter that are loaded into Python separately and merge into a single data frame. The final data contains a total of 139,919 loans with 145 columns of information. The information in the columns include information such as: loan type, amount, dates, and personal data on the credit history of the customer as well as the stats of the loan. The specific descriptions on each of the 145 columns is in: LCDataDictionary.xlsx.

Data exploration, cleaning and visualization

An initial exploration of the data resulted in two obvious problems that need to be addressed:

a. There are text descriptions of the data in the last rows of the id column (id column is empty since it is personal information). These cells were removed after sorting the data and deleting the last 8 rows. This procedure resulted in 139,311 loans.

b. There are many cells with NaNs. Some columns have no information at all, but are filled with NaNs. These columns were identified and removed with a custom function. The function counts the total number of NaNs in the colum and then remove columns with less NaN threshold. Initially reducing the number of columns to 142 and then to 36, maintaining the number of NaN per column to a maximum of 10. These leave only three columns with NaN zip_code, annual_inc, and last_credit_pull_d. Nothing further was done with the NaN as zip_code was remove (See below) and after filtering the loans, and other characteristics, the NaNs from the other columns were removed.

Table 1. Descriptive statistics for the feature amount loan from the 2017 Lending Club dataset. These measures correspond to the variable, before feature selection and elimination of "Current" loans.

Measure	Value
Count	139311
Mean	13707.77
Standard deviation	9006.37
Minimum	500
25%	6625
50%	12000
75%	19200
Maximum	40000

The total number of loans offered in 2017 was 139,311 with an average amount of US $ 13,707.77 +/- 9,006.37, a maximum pf 40,000 and a minimum of 500. Additional descriptors for this variable are presented in table 1 and for other features are documented in the notebook accompanying this report (Problem_loan_prediction_code.ipynb).

Feature selection

Target feature

The target feature corresponds to loan_status. There are nine different classes of loans, these can be grouped into three main categories:

- *Current loans* do not show problems, payments are current and for this reason were removed from the dataset.

- *Fully paid* loans are those that have been paid in full. It is not clear whether they were paid on time or not, but the assumption is that they did not show problems or delays before payment.

- *Problem loans* are all loans that have a problem, including Default, Grace Period, Do not meet credit policy, and/or Charged off.

Fully paid and problem compromise 58,006 loans.

Predicting features

The other 35 features (excluding status_loan) were classified into: non-numeric/categorical and numeric features.

There are *18* non-numerical variables. From these, zip_code, sub_grade, and initial_list_status, were removed. Data in zipcode included only 877 of the 43,000 in the USA and the information they may provide to the model should be contained in the state column. Similarly, subgrade is a subcategory of grade and initial_list_status is not clearly described.

Interest rate (int_rate) and employment length (emp_lengh) had string characters in most of the c ells. These characters were replaced or removed where necessary and the columns converted into a numeric variable. For example, the format in int_rate was: 7.49%. In this column, “%” was rem oved and the cell and column, became numeric.

Finally, two columns, issued date (issue_d) and last credit check (last_credit_pull_d) have date information. Both of these columns are formatted as “Month-year”. This data was converted into delta time by calculating the time in days from to the last date in the data set. After this transformation, both variables became numeric.

After processing the abovementioned columns, the dataset contains 11 categorical features All features are more or less well represented across the two categories (Fully paid and Problem loans) of the loan status variable (Figure 2).

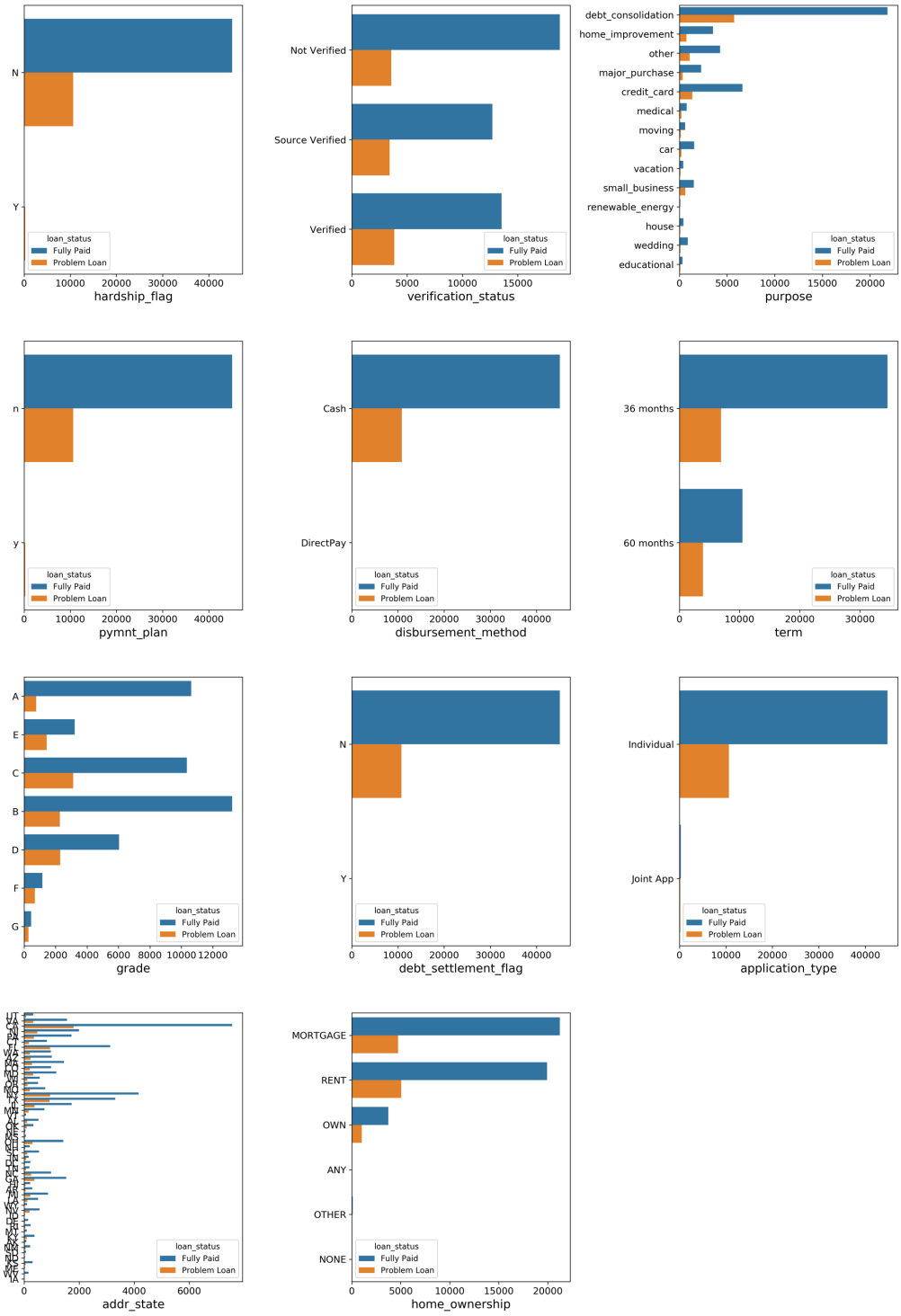

Figure 2. Bar plots showing the counts of fully paid and problem loans for each category in the categorical variables from the 2017 first two quarters of data.

The original number of numeric variables was 17, but with the conversion of categorical variables, this number increased to 21.

Of the numeric columns, policy_code had a single value (1). Installment, funded_amnt, funded_amnt_inv, and out_prncp_inv corresponded to similar types of data (loan_amnt and out_prncp – Table 2). These 4 features were removed.

Table 2 Number of unique observations for each of the numeric variables in the 2017 loan dataset.

Numeric feature	*No. of unique values in the feature*
loan_amnt	1208
revol_bal	26976
out_prncp	3447
out_prncp_inv	3460
total_pymnt	55342
total_pymnt_inv	52965
total_rec_late_fee	2627
recoveries	4976
collection_recovery_fee	2889
last_pymnt_amnt	47818
policy_code *	1
total_rec_int	48546
total_rec_prncp	11752
funded_amnt	1266
int_rate	420
installment	19462
emp_length	11
issue_d	6
funded_amnt_inv	9307
annual_inc	6415
last_credit_pull_d	124

A correlation analysis between numerical values did not indicate any concerning correlation between any pair of variables (Figure 3). Only total_rec_prncp showed higher correlations with total_payment ($r^2 = 0.97$) and total_paymt_inv ($r^2 = 0.95$ - Figure 3). All other correlations showed an r^2 lower than 0.80.

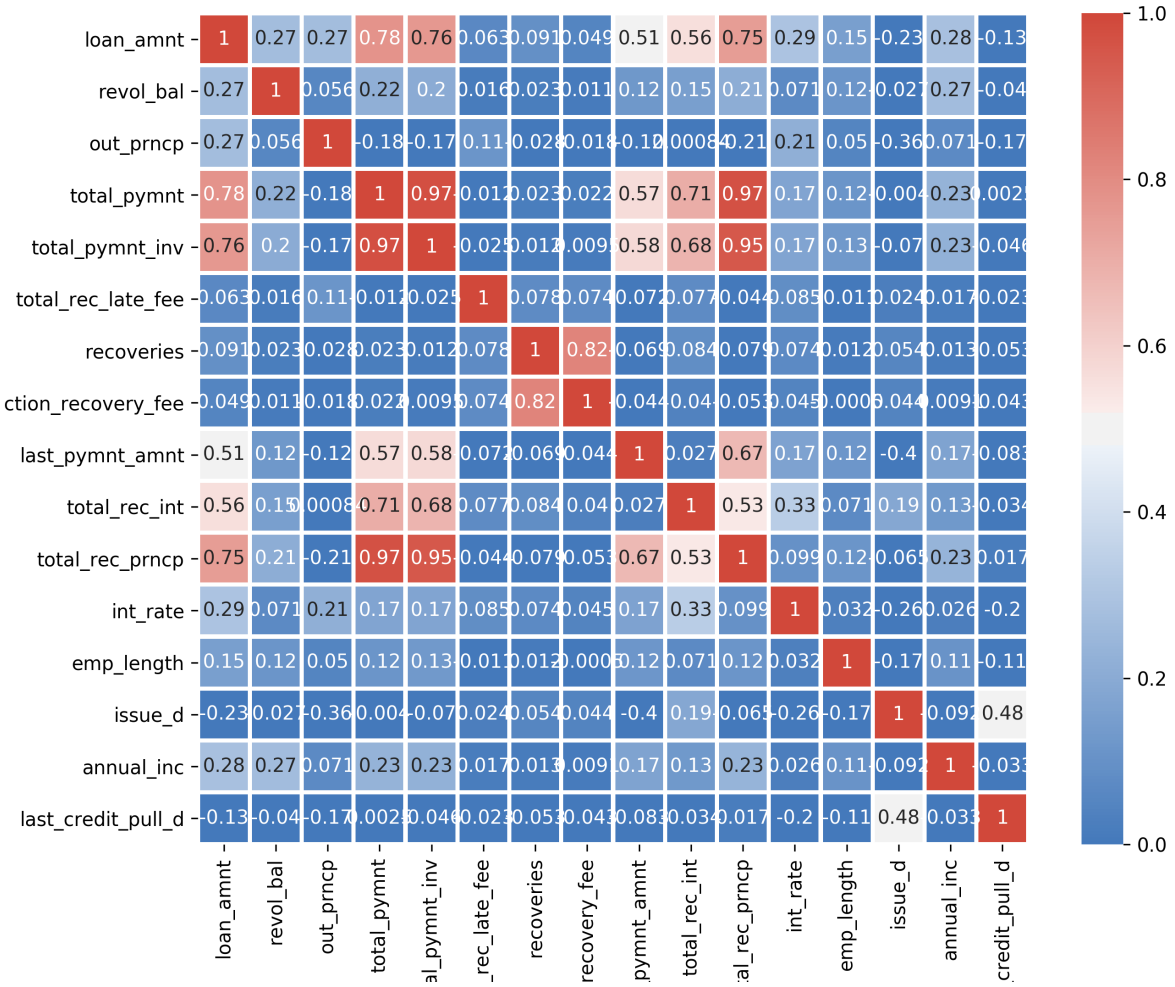

Figure 3 Correlations between numerical variables in the 2017 Lending Club dataset.

A simple boxplot visualization of the numeric variables showed the presence of outliers in most of them (Figure 4 - top).

Outlier analysis using the 1st and 3rd percentile revealed 8383 rows with data identified as outliers across all columns. These data points were eliminated. Loans with outliers on fewer columns were kept in the dataset. After this step, the number of loans in the dataset is 47,505 with 28 features.

Finally, the 11 categorical variables were encoded using LabelEncoder from scikit-learn, loan_status was also encoded. This is necessary for the neural network to work, as it does not take categorical variables.

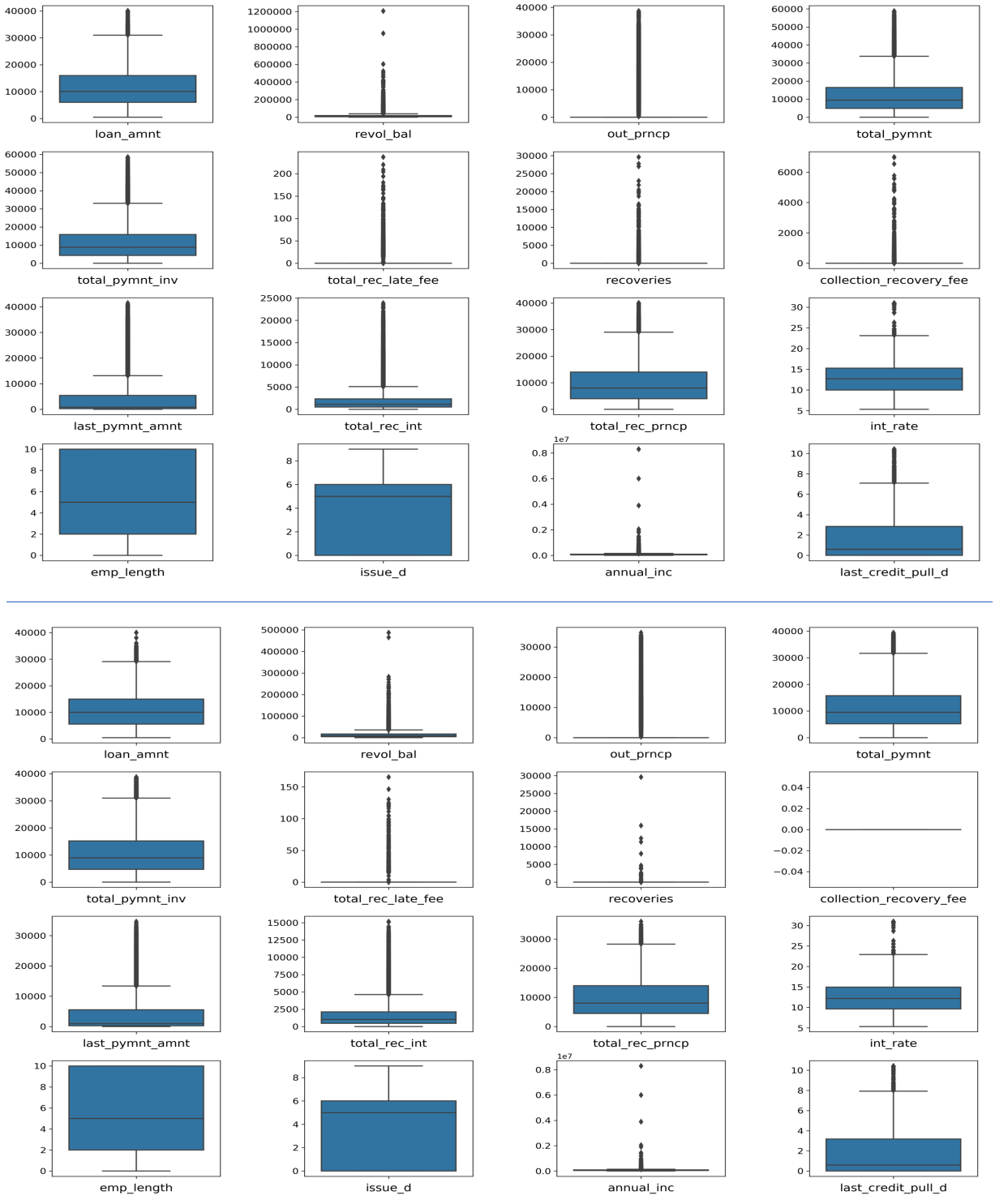

Figure 4 Boxplots of the numeric variables before (top) and after (bottom) outlier removal.

Algorithms and Techniques

Artificial neural networks (ANN) are machine learning algorithms based on the nervous system. Each network contains layers that have interconnected neurons each one processing the data and producing an output which is passed to the next layer (Gulli and Pal, 2017). This sequence of operations has two important properties: backpropagation and non-linear activation. Backpropagation updates the model based on the learning rate and the output loss, while non-linear activation allows the neural network to adapt to non-linear situations (McClure, 2017). For these reasons and the resistance to unbalance datasets, ANN are optimal for this project.

A simple ANN possesses a minimum of three layers: *Input*, receives the information. *Hidden*, process the data and transform the input for the output layer. It is in between the input and the output layer and can be scalable as the network can have as many hidden layers as necessary. 3. *Output* produces the final result (https://hackernoon.com/overview-of-artificial-neural-networks-and-its-applications-2525c1addff7).

Among the different parameters used on ANN architectures, the activation function, the optimizer, and the hyper parameter epoch are used to fine tune the model and provide the best prediction possible. The activation function is responsible for the activation, or firing, of each neural on a layer therefore allowing for the different layers to recognize the output from the previous later (https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0). There are many activation functions, one of the most commonly used is the Relu activation, which basically returns an activation of x for any positive x values and a 0 for all other values in the neuron (Relu activation formula: $A(x) = \max(0, x)$). This activation allows the network to be lighter as it makes the neurons activations sparse but efficient. However, it can make the network passive as the gradient tend to 0 due to the horizontal nature of the activation.

The sigmoid function, a nonlinear smoothing function is commonly used in the output layer as it can only output values between 0 and 1 (McClure, 2017). A neuron activated with the sigmoid function behaves similar to a perceptron, but allow for 'intermediate values' to be interpreted facilitating predictions (Gulli and Pal, 2017).

There are various neural network methods to minimize the loss function and improve the self-generated the internal learnable parameters (https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f). Optimizers are classified into *First* and *Second* order methods. Adam optimizer is a second order optimizer that includes a velocity component in addition to the gradient descent (Gulli and Pal, 2017). Additionally, Adam maintains separate learning rates for each weight and combines elements from other optimizers (Momentum and Adagram) making it memory efficient (https://smist08.wordpress.com/tag/adam-optimizer/).

Sequential models are improved by allowing the algorithm to go through the training set multiple time (McClure, 2017). Passes over the training data set by the model are called epochs. On each epoch, the neural network is exposed to every record in the dataset (https://deeplearning4j.org/glossary). Epochs allow the neural network to learn more about the

data, and in theory improve the model. However, improvements come to a cost in time and computational resources. While allowing the model to read the data more times, increasing the number of epochs may not necessary results in any gain in performance (Gulli and Pal, 2017).

The proposed ANN classifier model consists of:

- one input layer
- one hidden layer
- one output layer.

All layers are regular densely-connected layers uniformly initiated. The input and hidden layers use the relu activation function with 14 units. The output layer is activated by a sigmoid function with a single unit.

The network compiler uses the adam function as optimizer, binary_crossentropy for the loss, and set the accuracy as the metrics for the model.

In terms of coding, the model is simple and easily coded. Most of the 'difficulties' in the process came from data preprocessing mostly due to few inconsistencies in the format of some of the columns, for example int_rate was expected to be a number, but it had string characters ('%') that make it a categorical variable. When comparing the model to other models, I try to use the 2017 dataset, but the way the models were set-up did not allow this. Some of the columns were missing or the formats were different. This were avoided by using the dataset used for the formulation of the kaggle models.

Benchmark model

There are a few methods that have used data sets from the Lending Club statistics site. Of these two will be used as benchmark models:

- **Kaggle-1** (https://www.kaggle.com/mina20/building-a-neural-net-to-predict-default/notebook)
- **Kaggle-2** (https://www.kaggle.com/gagrawal/neural-net-with-keras)

This used neural network for the predictions, kaggle-1 uses tensorflow directly and kaggle-2 uses keras and tensorflow. Neither report metrics, but after running them here (See below), they have an accuracy of 94.47 and 93.0% respectively. In addition to running the model and estimating the accuracy, the F-score will also be calculated with the predictions of these models.

METHODOLOGY

Data preprocessing

In order to prepare the data for the model, it was split into the target variable (loan_status) and the predictors (features). Then these portions are divided into training (75%), validation (20%), and test (5%) subsets.

The features were standardized using StandardScaler which removes the mean and scales the data to unit of variance. This step is necessary to prevent bad behaviors in the model particularly from features that are not normally distributed.

Implementation

The proposed model was built in keras using TensorFlow backend, using the following code:

- Initiation. Start the network and set the environment to sequential.

  classifier = Sequential()

- Input layer. Takes the data in the correct input and transform it to be used in the hidden layer.

  classifier.add(Dense(units = 14, kernel_initializer = 'uniform', activation = 'relu', input_dim = 27))

- Hidden layer. Determines the activation status, inputs the weight and bios from the input layer and outputs that information for the output layer. This model only has one hidden layer.

  classifier.add(Dense(units = 14, kernel_initializer = 'uniform', activation = 'relu'))

- Output layer. Takes the output from the hidden layer and perform calculations to determine the output value for each data point.

  classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

- Compilation. Set the hyperparameters for the model.

  classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

- Training / fitting. Train the model with the training dataset.
  classifier.fit(X_train, y_train, batch_size = 10, epochs = 10)

Accuracy of the model and loss of the training set is measured during training.

Then the model will be validated in two ways:

- Cross-validation using the training set. The classifier will be run using a kfold (cv) of 10. Basically, split the data into 10 subsets and run it several times making the evaluation set one of the folds on each run, effectively training the model with different portions of the data. The accuracy of the cross validation will be the average accuracy of calculated on each run.

  ```
  classifier_cv=KerasClassifier(build_fn=build_classifier, batch_size =10, epochs = 10)

  accuracies=cross_val_score(estimator=classifier_cv, X=X_train, y = y_train, cv = 10)
  ```

- Validation using the validation set. Use the validation subset to determine the accuracy of the neural network on a data different to the training set.

  ```
  loss, accuracy = classifier.evaluate(X_val, y_val,batch_size=128, verbose=0)
  ```

Once the model has been validated, it will be tested on the test data. The predictions from the test run will be used to determine accuracy and F-score of the model. To compare the current model with the benchmark models, the 2007-2015 data used by the kaggle models will be used.

- Predict the target feature on the test set:

  ```
  y_pred = classifier.predict(X_test)

  y_pred = (y_pred > 0.5)
  ```

- Determine the accuracy

  ```
  loss, accuracy = classifier.evaluate(X_test, y_test,batch_size=128, verbose=0)
  ```

- Make the confusion matrix and calculate the F-score:

  ```
  cnf_matrix = confusion_matrix(y_test, y_pred, labels=[0, 1])

  f1_score(y_test, y_pred.round(), average='binary')
  ```

RESULTS

Model evaluation and validation

After compiling the model, it was fitted with the training data set, run for 10 epochs with a batch size of 10. This initial run had an accuracy of 99.74% and a loss of 0.0093. The cross validation of the model using a k-fold of 10 returned an average accuracy of 99.65% +/- 0.15 and the validation with the validation subset was 99.6913% accurate with only 0.0118 loss.

These high accuracies might be the result of overfitting, even though they were already tested on different subsets. However, as a precaution the classifier was tested on data from the 4th quarter of 2016 (see notebook 2016_pred.ipynb). This test resulted was of 98.49% (20,005 loans) accurate. The model was also tested in a dataset that included loans from 2007-2015 and is compiled in kaggle (https://www.kaggle.com/wendykan/lending-club-loan-data). In this larger dataset, the model showed an accuracy of 99.82% (see notebook 2007-2015_pred.ipynb). It is very unlikely that the model is over fitted given that all datasets are different.

Model optimization

An optimization of the model using grid search that iterates through different parameters suggested to change the model by adding 10 extra epochs. The final model was:

- Initiation:
  classifier = Sequential()

- Input layer:
  classifier.add(Dense(units = 14, kernel_initializer = 'uniform', activation = 'relu', input_dim = 27))

- Hidden layer:
  classifier.add(Dense(units = 14, kernel_initializer = 'uniform', activation = 'relu'))

- Output layer:
  classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))

- Compiler:
  classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

- Training:
  classifier.fit(X_train, y_train, batch_size = 10, epochs = 20)

The model with 20 epochs resulted in the same accuracy of 99.74% on the validation dataset.

Implementation

The optimized classifier on the test set had an accuracy of 99.72% and an F-score of 0.9871. It only misclassified 43 loans out of the 11,877 loans (Figure 5).

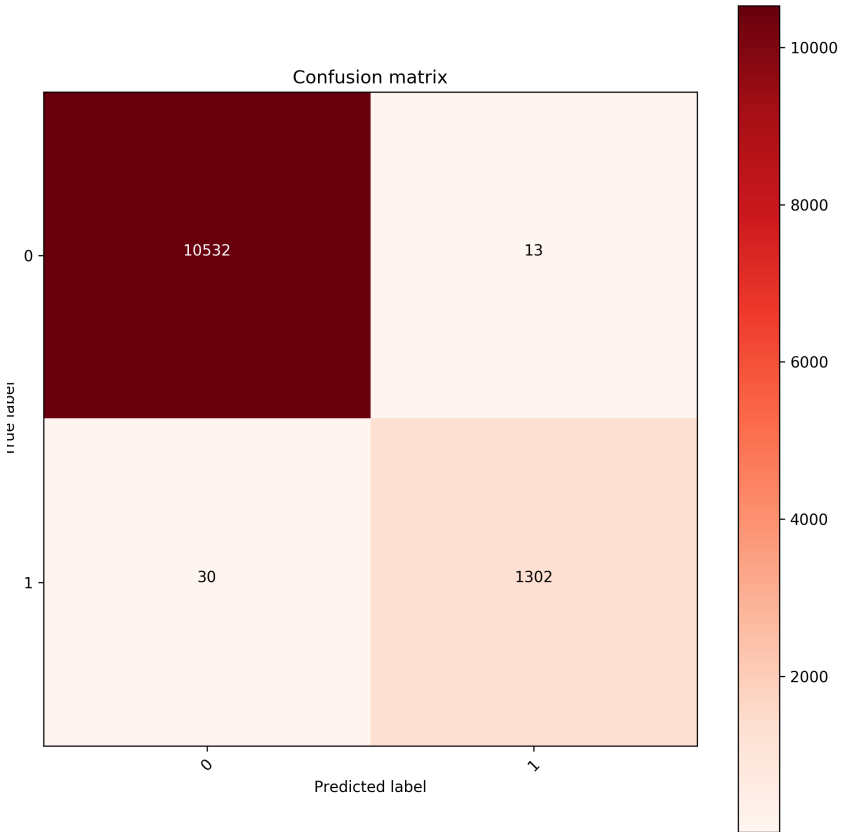

Figure 5. Confusion matrix of the 2017 dataset predictions. Paid loans as 0 and Problem loans as 1.

Comparison with benchmark models

The proposed ANN classifier was fitted with the date from 2007-2015 as to not to change the kaggle models to use the 2017 dataset. Kaggle-1 and kaggle-2 predict default loans on Lending Club data from 2007 to 2015 and both also use neural networks as their preferred method.

Kaggle-1 uses tensorflow with a DNNRegressor on 78 features and 37,379 loans. The original script only reports the loss during fitting with a value of 0.0253. The measured accuracy (number of correct predictions over total number of predictions) was 94.47%. The precision was 0.9447 and the recall 1. This model was not able to identify *problem loans*; therefore, it was not possible to determine the F-Score (Table 3, Figure 6A).

Kaggle-2 uses keras on tensorflow with a 3-layer classifier network and 68 features on 1,000 loans. The loss during fitting was 1.4029 and the testing accuracy 91.2%. The precision was 0.93 and the recall 1. This model also failed to predict *problem loans*, and the F-Score was not determined (Table 3, Figure 6B).

Table 3 Summary table with the comparisons between the proposed model and the models from two models uploaded in kaggle.

	Kaggle-1	***Kaggle-2***	***ANN Proposed here***
Model	DNN Regressor	Classifier 3 layers	Classifier 3 layers
Library	Tensorflow	Keras-tensorflow	Keras-tensorflow
No. loans tested	37,379	1,000	97,060
No. features	78	68	23
Loss during fitting	0.025	1.12	0.01
Accuracy on test dataset	94.47	93.0	97.30
Precision	0.9447	0.93	0.969
Recall	1	1	0.999
F-Score	Did not predict Problem loans 1 in Figure 6A*	Did not predict Problem loans 0 in Figure 6B*	0.9058

* The difference in the code correspond to original coding of the feature.

The ANN model proposed here is also based on keras and tensorflow with 3-layers, but it uses 23 features and 97,060 loans for testing. This model has an accuracy of 99.78%, loss of 0.0119 during fitting, and predicted both problem and paid loans with an F-Score of 0.9058 with an accuracy of 97.30% (Table 3, Figure 6 C).

CONCLUSION

Free-visualization

Figure 5 shows the main take away from this project. It is a confusion matrix that summarizes the main results of the project. In this figure, it is clearly shown how the model is capable of predicting both *paid (0)* and *problem loans (1).* The number of correctly classified loans is high (11,834) and only a small portion of the loans (43) were misclassified. This information combined with the information on figure 6 shows how powerful the solution presented here is in identifying problem loans in several datasets.

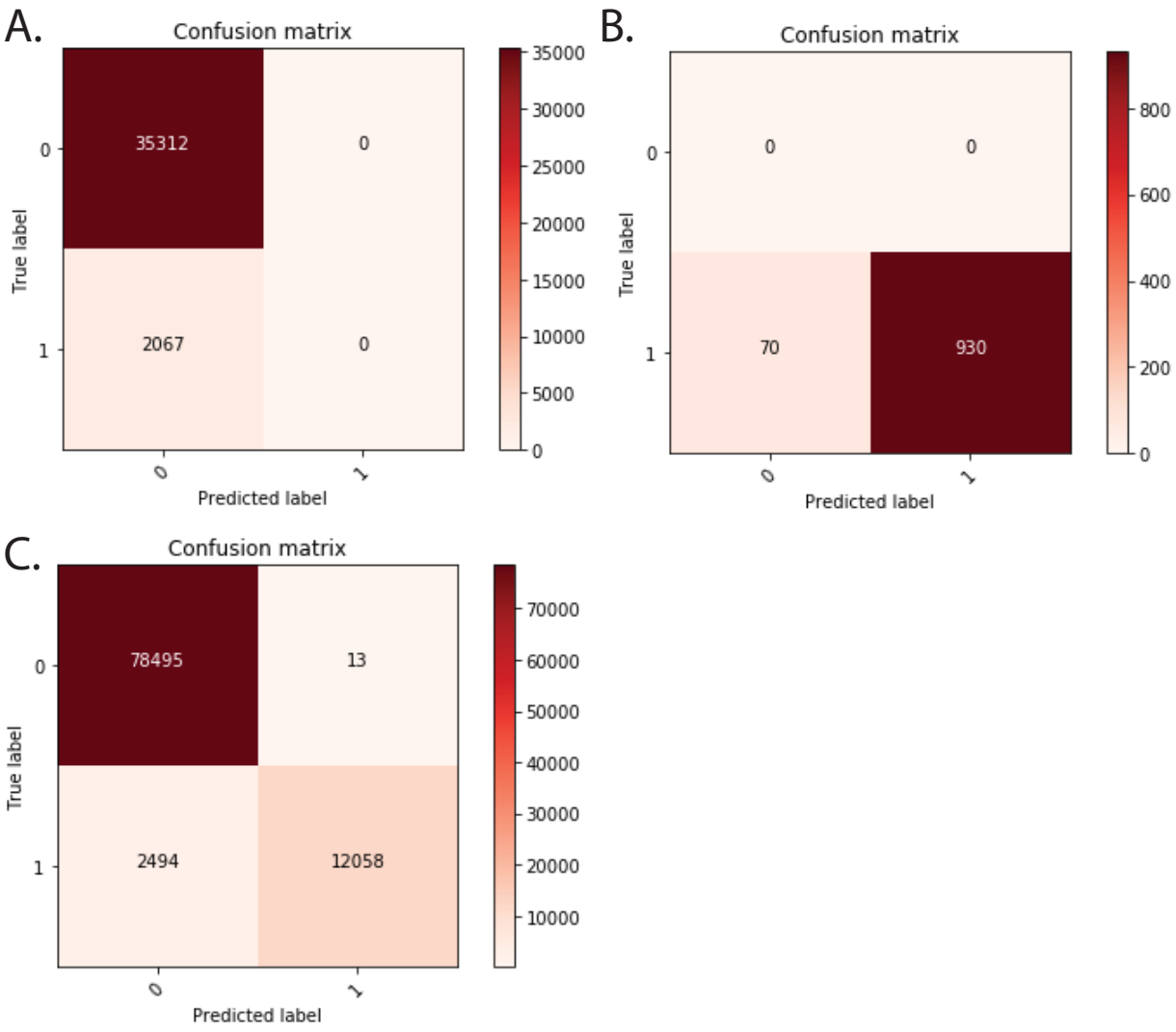

Figure 6 Confusion matrices for kaggle-1 (A), kaggle-2 (B), and the ANN proposed in this project (C) using the 2007-2015 dataset. In A and C, Paid loans as 0 and Problem loans as 1. In B is the opposite.

Reflection

The proposed model outperforms other methods by 5-7 points in terms of accuracy and predicts both categories. The outperformance could be the result of:

a. Cleaning data

b. Labeling of the categorical features.

c. The number of selected features.

d. The standardization of the features before the model.

Kaggle-1 uses a different structure to that in the current model, making the comparison slightly difficult. On the other hand, kaggle-2 and the ANN proposed here have the same 3-layer structure but the labelling is different, and in kaggle-2 the number of selected features is higher, and it lack feature standardization. Combined these differences may account for the differences in accuracy seen in the models. Models using traditional machine learning algorithms result in lower accuracies, as shown by at least one project that used R and several models to predict problem loans in a dataset from the Lending Club (https://rstudio-pubs-static.s3.amazonaws.com/203258_d20c1a34bc094151a0a1e4f4180c5f6f.html).

Improvement

With accuracies of 99.5% and larger and F-Scores bordering 0.98, it is difficult to come up with an improvement for the model. Rather it will be interesting to:

- Add more features and determine how the accuracy is affected.

- Test this model in datasets other than those from the Lending Club.

REFERENCES

Gulli, A. and S. Pal, 2017. Deep Learning with Keras. Birmigham – Mumbay. Packt. 296p.

McClure, N. 2017. TensorFlow machine learning cookbooj. Birmigham – Mumbay. Packt. 851p.

Ren, J. 2012. ANN vs. SVN: Which one performs better in classification of MCCs in mammogram imaging. Knowledge-Based Systems. 26, 144-153.