ICAT 3120. Machine Learning. Final Report
Risk management in financial services: an overview of a credit score model using ML.
Reyver Serna: 119652

Introduction

Banks and other financial institutions often receive plenty of loan applications rom their customers. The high degree of uncertainty present in these applications represents a challenge for these organizations because they must put a lot of resources to predict if the customer will be able to pay the loan back. For that reason, banks and other lenders base their decisions in the customer's aspects such as past transactions, income, savings, assets… A customer is low risk if the customer has a history of paying back previous loans. On the contrary, she is labelled as a high-risk if she has defaulted some of her duties.

When analysing the data, these organizations need to learn the type of class customer in order to take the best decision when a new application arrives. Thanks to the development of Machine Learning, ML, in the area of banking and financial services, nowadays these organizations can build credit risk models that predict the credibility of a customer automatically based on observable data. According to Alpaydin (2014) financial institutions find themselves in a need for accurate and effective models that lead them to make decisions in order to increase the potential gain when dealing with different loan applications. This need comes from past mistakes that lead to global financial in 2007- 2008.

In the following pages, I will offer a summary overview of a Machine Learning application based on a credit score model that seeks to predict if a customer applying for a loan is default (bad) or good. The aim of this credit score application based on a Logistic Regression model seeks to avoid misclassifications (wrong decisions) that may result in high costs for the lending institution.

Theory behind credit risk: Probability Theory

Most of the credit risk model boosted by Machine Learning tools are based on probability theory. Generally, probability theory is defined as the mathematical framework used to represent uncertain statements (Goodfellow, Bengio & Courville, 2016). Concretely, probability theory serves with the ways to quantify uncertainty and build new uncertain statements. Goodfellow, Bengio & Courville (2016), state that in Machine Learning and Artificial intelligence applications, probability theory is mainly used for two things: First, to tell us how the system should reason and why the design of the algorithms should compute approximations of them. Second, to analyse the behaviour of the system by using probability and statistics theories.

In ML, probability theory is used strongly since it must always deal with uncertain quantities and non-deterministic quantities (these are also called *stochastic*). These two quantities come from many sources, therefore the ML system and the expert need to know how to reason in the presence of uncertainty. In other words, the human and the system need to think beyond the mathematical statements due to the difficulty to think that any proposition is absolutely true or guaranteed to happen.

When working with probability theory in ML, it is important to remember three facts: Firstly, "the inherent stochasticity in the system being modelled" (Goodfellow, Bengio & Courville, 2016). Secondly, that the incomplete observability of the events is a given. Despite the system being deterministic by nature, we cannot observe all the variables that guides the behaviour of the system. Thirdly, the model can be incomplete since, sometimes, it should discard some of the observed information. Thus, three premises are inherent in the principle that states: "it is more practical to use a simple but uncertain rule rather than a complex but certain one" (Elmusrati, 2021).

Classification and the Bayes' rule

A more concrete theoretical instance of my credit risk model lies in the inferences given by the Bayes' rule for classification. As discussed before, the probability theory gives us the framework to make decision under uncertainty. On the other hand, the Bayes' rule provides us with tools to calculate the probability of classes that help the expert to make rational choices and reduce the expected risk (Alpaydin, 2014, p. 49). In other words, how the expert can measure those extra pieces of knowledge that she does not have access to (also called, *unobservable variables*). The Bayes' rule is written as:

$$\mathrm{P(C|x)} = \frac{P(x|C)P(C)}{\mathrm{p(x)}}$$

In this model the credibility of the customer will be determined by the Chi-Square distribution in order to denote the different classes of customers. This is a type of random variable that works pretty well with independent variables.Alpaydin (2014) defines Random Variables as "a function that assigns a number to each outcome in the sample space of a random experiment." When solving the problem using the Bayes' rule, the model calculates the data in three process: *prior probability*, *class likelihood* and the *posterior probability*.

Prior probability is the knowledge of having to the value C before looking at the observables x. prior probability P(C=1) will take value 1 corresponding to a customer being high-risk regarding of the variables x:

$$P(\mathrm{C} = 0) + P(\mathrm{C} = 1) = 1$$

Class likelihood. P(x|C) is the class likelihood. It corresponds to the conditional probability that an event belonging to C has the associated observations value x. For instance, in the cases that $p(x_1,x_2|C=1)$ is the probability that a high-risk customer has her $X_1 = x_1$ and $X_2 = x_2$ regarding what the data tells us regarding the class. "p(x), the *evidence*, is the marginal probability that an observation x is seen, regardless of whether it is a positive or negative example" (Alpaydin, 2014).

$$p(\boldsymbol{x}) = \sum p(\boldsymbol{x}) = p(\boldsymbol{x},\mathrm{C}) = p(\boldsymbol{x}|\mathrm{C}=1)P(\mathrm{C}=1) + p(\boldsymbol{x}|\mathrm{C}=0)P(\mathrm{C}=0)$$

The posterior probability is calculated by combining the prior probability and the class likelihood by using the Bayes' rule . Namely, P(C|x) after seeing the observation x.

$$posterior = \frac{prior\ x\ likelihood}{evidence}$$

Therefore, the Bayes' classifier chooses the class with the highest posterior probability. In other words, the expert chooses C_i if P(C|x) = max P(Cx|x)

Methodology: creating the Credit Risk Model using Python.

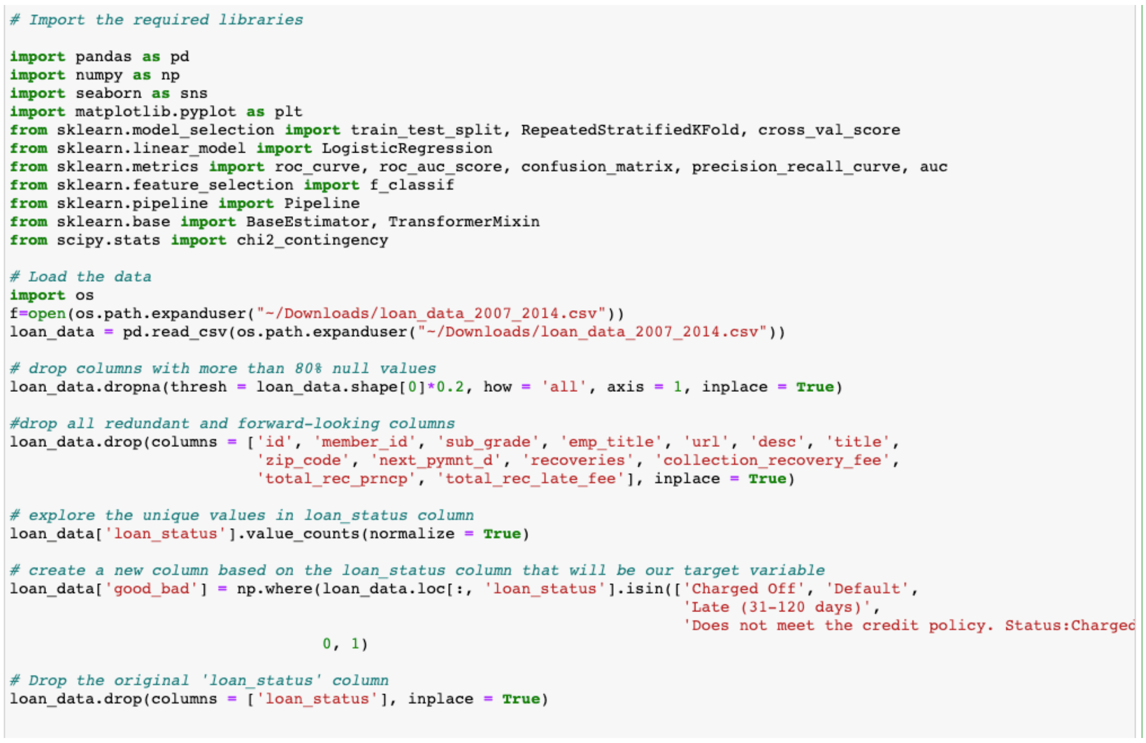
In this section I will replicate and explain a statistical credit risk model which that predicts the probability of credit default. This is a modified replica of the model created by Asad Mumtaz (Mumtaz, 2020a), a finance professional interested in data analytics and machine learning (See the link in the reference page).This model can be classified as a supervised method since it uses Logistic Regression for making predictions. For this, Python will be used to program build the model thanks to wide varieties of libraries that can help to build the model (Pandas, NumPy, Seaborn, Scikit, Matplot…). Also, a credit scorecard will be built in order to implement the prediction model.

This section offers a summarised version of the whole process due to the length of the process. However, the most important is that the reader grasps a wholistic picture of the main parts implied when creating a supervised model that works with large datasets. A separate Jupyter notebook (ipynb) will be attached to this report. So that the reader can see the whole code with comments due to

For this task, I will use the dataset from Lending Club, a US P2P lender. The raw data displays the information of over 450,000 consumer loans issued between 2007 and 2014 with approximately 75 features. This data can be found In Kaggle in the following link: https://drive.google.com/file/d/1xaF743cmUgI5kc76I86AeZDE84SMkMnt/view. When working with this data, some features will be deleted in order to increase the accuracy of the model. Concretely, features with more than 80% of missing values, features not related to credit-risk, and some other forward-looking after the borrower has defaulted, will be removed.

Data Exploration

To start with, all the necessary libraries will be imported to the systems in order to have the tools that enable our statistical model. Also, the data will be loaded to the system and cleaned as mentioned before. In addition, a target variable will be created in order to set the classification terms (Default = 0; Good=1).

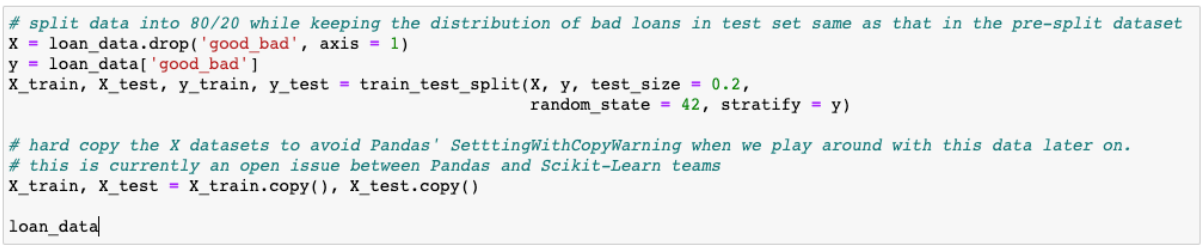


```python
# Import the required libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, RepeatedStratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix, precision_recall_curve, auc
from sklearn.feature_selection import f_classif
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from scipy.stats import chi2_contingency

# Load the data
import os
f=open(os.path.expanduser("~/Downloads/loan_data_2007_2014.csv"))
loan_data = pd.read_csv(os.path.expanduser("~/Downloads/loan_data_2007_2014.csv"))

# drop columns with more than 80% null values
loan_data.dropna(thresh = loan_data.shape[0]*0.2, how = 'all', axis = 1, inplace = True)

#drop all redundant and forward-looking columns
loan_data.drop(columns = ['id', 'member_id', 'sub_grade', 'emp_title', 'url', 'desc', 'title',
                          'zip_code', 'next_pymnt_d', 'recoveries', 'collection_recovery_fee',
                          'total_rec_prncp', 'total_rec_late_fee'], inplace = True)

# explore the unique values in loan_status column
loan_data['loan_status'].value_counts(normalize = True)

# create a new column based on the loan_status column that will be our target variable
loan_data['good_bad'] = np.where(loan_data.loc[:, 'loan_status'].isin(['Charged Off', 'Default',
                                                                       'Late (31-120 days)',
                                                                       'Does not meet the credit policy. Status:Charged
                                 0, 1)

# Drop the original 'loan_status' column
loan_data.drop(columns = ['loan_status'], inplace = True)
```

Image 1: data exploration

Data Stratification: splitting the data.

Next, the data will be spitted in two parts: training set and test set. According to Mumtaz (2020), the division will be 80% for the training set and 20% for the test set. In this case, the code also performed a Repeated Stratified K-fold testing on the training set to pre-evaluate the model. In more practical words, this will stratify the data so that "the distribution of good and bad loans in the test set is the same as that in the pre-split data" (*Ibid*). This can be done by the *train_test_split* function with the *stratify* parameter (Image 1: data exploration).

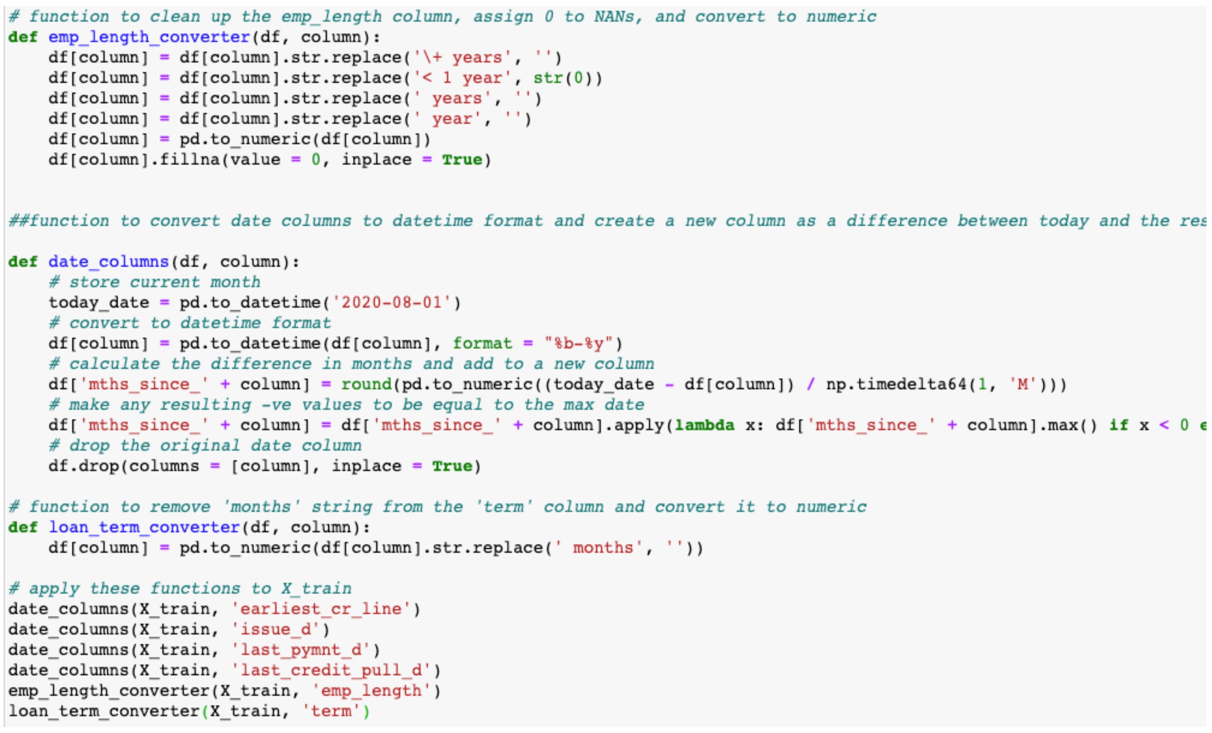


```python
# split data into 80/20 while keeping the distribution of bad loans in test set same as that in the pre-split dataset
X = loan_data.drop('good_bad', axis = 1)
y = loan_data['good_bad']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                    random_state = 42, stratify = y)

# hard copy the X datasets to avoid Pandas' SettingWithCopyWarning when we play around with this data later on.
# this is currently an open issue between Pandas and Scikit-Learn teams
X_train, X_test = X_train.copy(), X_test.copy()

loan_data
```

Image 2: data stratification

Data Cleaning

The following code-block will remove text from the columns and convert into numeric. Concretely, from *emp_length* and *term* columns. In addition, the dates will be converted to Python *datetime* format.



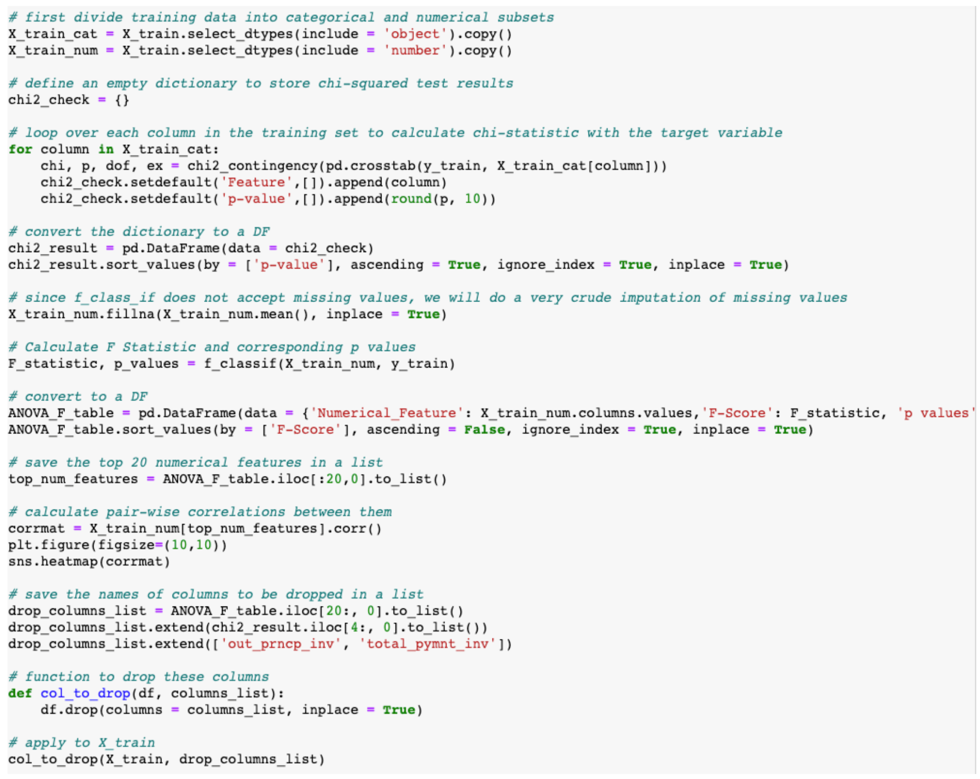
```
# function to clean up the emp_length column, assign 0 to NANs, and convert to numeric
def emp_length_converter(df, column):
    df[column] = df[column].str.replace('\+ years', '')
    df[column] = df[column].str.replace('< 1 year', str(0))
    df[column] = df[column].str.replace(' years', '')
    df[column] = df[column].str.replace(' year', '')
    df[column] = pd.to_numeric(df[column])
    df[column].fillna(value = 0, inplace = True)


##function to convert date columns to datetime format and create a new column as a difference between today and the res

def date_columns(df, column):
    # store current month
    today_date = pd.to_datetime('2020-08-01')
    # convert to datetime format
    df[column] = pd.to_datetime(df[column], format = "%b-%y")
    # calculate the difference in months and add to a new column
    df['mths_since_' + column] = round(pd.to_numeric((today_date - df[column]) / np.timedelta64(1, 'M')))
    # make any resulting -ve values to be equal to the max date
    df['mths_since_' + column] = df['mths_since_' + column].apply(lambda x: df['mths_since_' + column].max() if x < 0 e
    # drop the original date column
    df.drop(columns = [column], inplace = True)

# function to remove 'months' string from the 'term' column and convert it to numeric
def loan_term_converter(df, column):
    df[column] = pd.to_numeric(df[column].str.replace(' months', ''))

# apply these functions to X_train
date_columns(X_train, 'earliest_cr_line')
date_columns(X_train, 'issue_d')
date_columns(X_train, 'last_pymnt_d')
date_columns(X_train, 'last_credit_pull_d')
emp_length_converter(X_train, 'emp_length')
loan_term_converter(X_train, 'term')
```

Image 3: data cleaning

Feature Selection

In the feature selection, the model will identify the most suitable features for the binary classification. In this case Chi-square distribution will be used for categorical features and ANOVA F-statistics for numerical features. In this aspect, Sayes et al. (2007) assert that feature selection has the objective, within manifold, to avoid overfitting and improve the model performance; to provide a faster and more cost-effective model; and gain a deeper insight of the data. Specifically, the author of the model chose Chi-squared test in order "to determine the extent of relationship or dependence between two categorical variables. In this case one categorical input feature, and the other, a categorical target variable (Mumtaz, 2020b).

In this model, there is also two blocks of code for setting the categorical variables: one for One-hot Encoding and the Weight of Evidence, WoE, and other for Binning and feature engineering. The former creates dummy variables of the four final categorical variables and update the test set. The later creates a new categorical feature for all numerical and categorical variables based on the WoE and Information Value (IV) to perform the feature engineering in order to differentiate between good (Good = 1) and bad customers (Default = 0). These two blocks of codes will not be explaining here since goes beyond the scope of this overall aim of this report. For more detail information about this, please check the Jupyter notebook attached to this report.



```python
# first divide training data into categorical and numerical subsets
X_train_cat = X_train.select_dtypes(include = 'object').copy()
X_train_num = X_train.select_dtypes(include = 'number').copy()

# define an empty dictionary to store chi-squared test results
chi2_check = {}

# loop over each column in the training set to calculate chi-statistic with the target variable
for column in X_train_cat:
    chi, p, dof, ex = chi2_contingency(pd.crosstab(y_train, X_train_cat[column]))
    chi2_check.setdefault('Feature',[]).append(column)
    chi2_check.setdefault('p-value',[]).append(round(p, 10))

# convert the dictionary to a DF
chi2_result = pd.DataFrame(data = chi2_check)
chi2_result.sort_values(by = ['p-value'], ascending = True, ignore_index = True, inplace = True)

# since f_class_if does not accept missing values, we will do a very crude imputation of missing values
X_train_num.fillna(X_train_num.mean(), inplace = True)

# Calculate F Statistic and corresponding p values
F_statistic, p_values = f_classif(X_train_num, y_train)

# convert to a DF
ANOVA_F_table = pd.DataFrame(data = {'Numerical_Feature': X_train_num.columns.values,'F-Score': F_statistic, 'p values'
ANOVA_F_table.sort_values(by = ['F-Score'], ascending = False, ignore_index = True, inplace = True)

# save the top 20 numerical features in a list
top_num_features = ANOVA_F_table.iloc[:20,0].to_list()

# calculate pair-wise correlations between them
corrmat = X_train_num[top_num_features].corr()
plt.figure(figsize=(10,10))
sns.heatmap(corrmat)

# save the names of columns to be dropped in a list
drop_columns_list = ANOVA_F_table.iloc[20:, 0].to_list()
drop_columns_list.extend(chi2_result.iloc[4:, 0].to_list())
drop_columns_list.extend(['out_prncp_inv', 'total_pymnt_inv'])

# function to drop these columns
def col_to_drop(df, columns_list):
    df.drop(columns = columns_list, inplace = True)

# apply to X_train
col_to_drop(X_train, drop_columns_list)
```

Image 4: feature selection

Model Training and Prediction Model

Here where the Machine Learning magic takes place. the logistic regression will be fitted into the training set and evaluate it using *RepeatedStratifiedKFold*. With the class weight parameter, the *LogisticRegression* will balance the classes, forcing the model to learn and penalize coefficients that are cost-sensitive (i.e: false negatives, false positives…). Area Under Receiver Operating Characteristics, AUROC, will be the evaluation metric (very common in credit scoring). A summary table will be created with the feature names and the coefficient returned from the model.

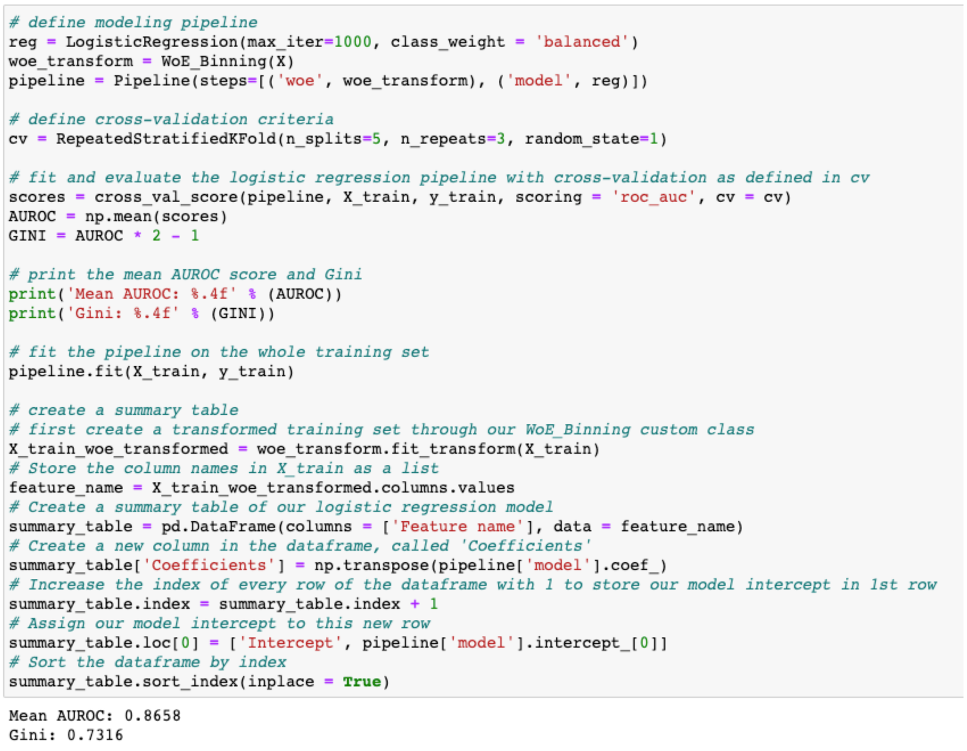
According to Müller & Guido (2017) Logistic Regression is one of the most commonly used algorithms to deal with classification problems. IMPORTANT! Logistic regression should not be confused with Linear Regression since this last one seeks to minimize the mean squared error between predictions.

To overcome the problem of overfitting the code uses another dataset, referred as the validation set. This is used to validate the prediction performance of the model and improve its generalization ability. This dataset can be provided in a separate manner or by taking some data from the training set (Rogers & Girolami, 2012).

In this model, the validation will be performed by using the cross-validation technique. This consists of the calculation of the loss from the validation set. This technique allows to make a more efficient use of the data.

Concretely, this technique will use *K-fold cross-validation* which "splits the data into *K* equally size blocks in the validation set. The loss will be calculated by averaging the *K* loss values in the final loss value. The model will be commanded to perform a stratified and repeated *K-fold cross-validation*. This simply implies the repetition of the cross-validation ultiple times and reporting the mean result across all folds from all runs. As a result, the model obtains a more accurate estimation of the mean as calculated in the standard error. (Rogers & Girolami, 2012).

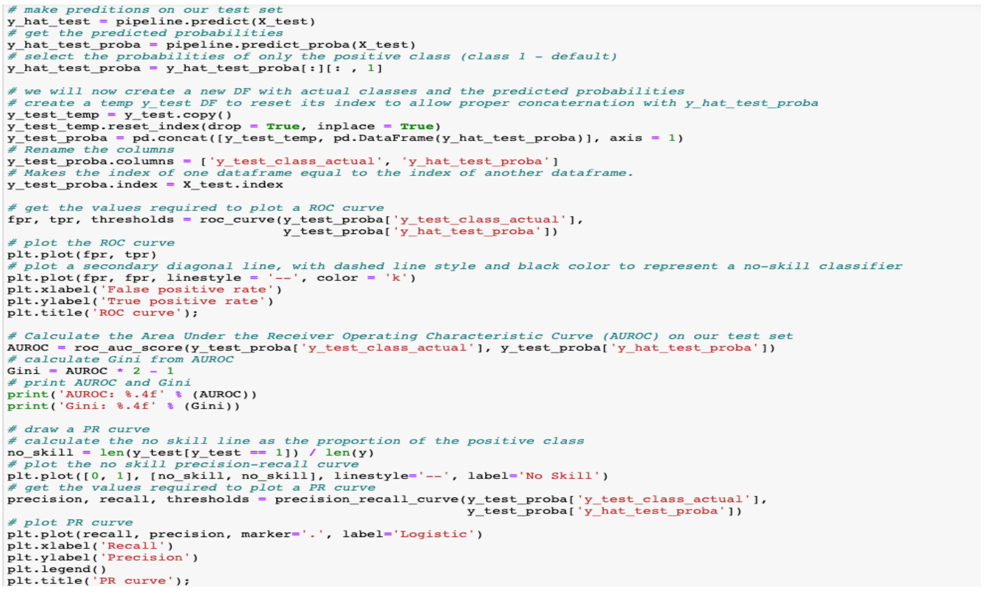
Once the model training is performing well, the logistic regression will be applied to predict the probability of default on the test set (next image).



```
# define modeling pipeline
reg = LogisticRegression(max_iter=1000, class_weight = 'balanced')
woe_transform = WoE_Binning(X)
pipeline = Pipeline(steps=[('woe', woe_transform), ('model', reg)])

# define cross-validation criteria
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=1)

# fit and evaluate the logistic regression pipeline with cross-validation as defined in cv
scores = cross_val_score(pipeline, X_train, y_train, scoring = 'roc_auc', cv = cv)
AUROC = np.mean(scores)
GINI = AUROC * 2 - 1

# print the mean AUROC score and Gini
print('Mean AUROC: %.4f' % (AUROC))
print('Gini: %.4f' % (GINI))

# fit the pipeline on the whole training set
pipeline.fit(X_train, y_train)

# create a summary table
# first create a transformed training set through our WoE_Binning custom class
X_train_woe_transformed = woe_transform.fit_transform(X_train)
# Store the column names in X_train as a list
feature_name = X_train_woe_transformed.columns.values
# Create a summary table of our logistic regression model
summary_table = pd.DataFrame(columns = ['Feature name'], data = feature_name)
# Create a new column in the dataframe, called 'Coefficients'
summary_table['Coefficients'] = np.transpose(pipeline['model'].coef_)
# Increase the index of every row of the dataframe with 1 to store our model intercept in 1st row
summary_table.index = summary_table.index + 1
# Assign our model intercept to this new row
summary_table.loc[0] = ['Intercept', pipeline['model'].intercept_[0]]
# Sort the dataframe by index
summary_table.sort_index(inplace = True)
```

```
Mean AUROC: 0.8658
Gini: 0.7316
```

Image 5: model training

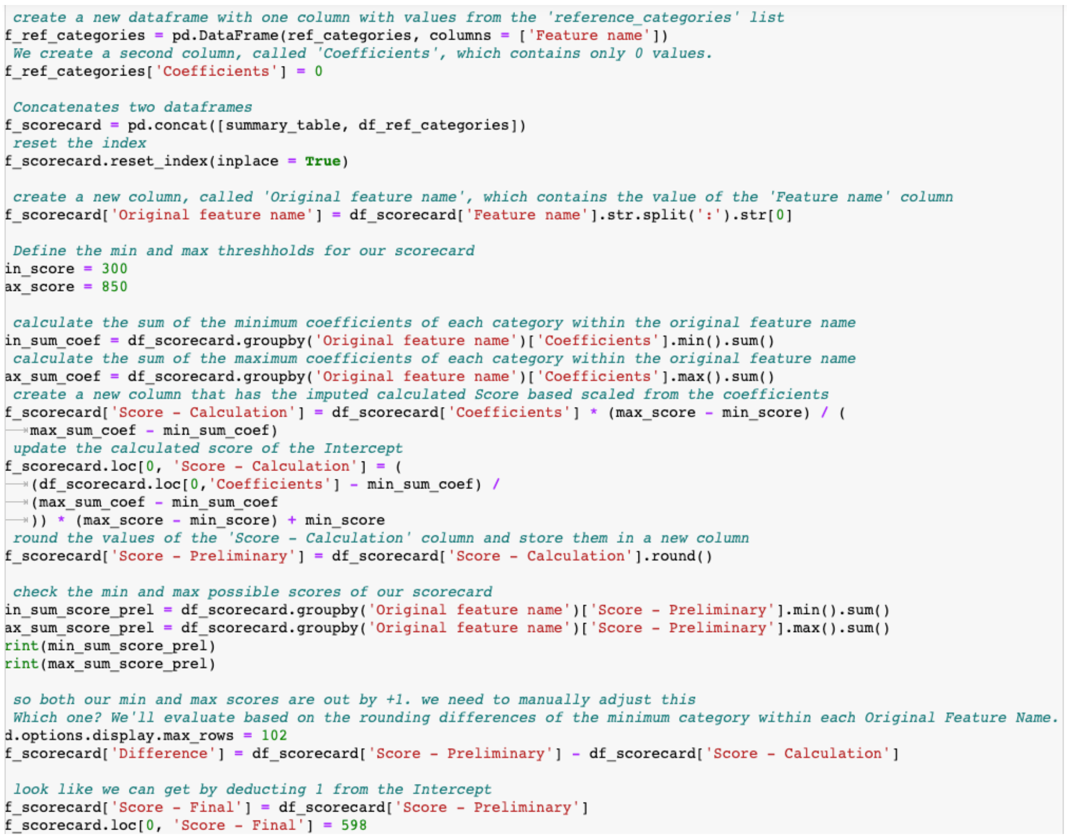


```
# make preditions on our test set
y_hat_test = pipeline.predict(X_test)
# get the predicted probabilities
y_hat_test_proba = pipeline.predict_proba(X_test)
# select the probabilities of only the positive class (class 1 - default)
y_hat_test_proba = y_hat_test_proba[:][: , 1]

# we will now create a new DF with actual classes and the predicted probabilities
# create a temp y_test DF to reset its index to allow proper concaternation with y_hat_test_proba
y_test_temp = y_test.copy()
y_test_temp.reset_index(drop = True, inplace = True)
y_test_proba = pd.concat([y_test_temp, pd.DataFrame(y_hat_test_proba)], axis = 1)
# Rename the columns
y_test_proba.columns = ['y_test_class_actual', 'y_hat_test_proba']
# Makes the index of one dataframe equal to the index of another dataframe.
y_test_proba.index = X_test.index

# get the values required to plot a ROC curve
fpr, tpr, thresholds = roc_curve(y_test_proba['y_test_class_actual'],
                                 y_test_proba['y_hat_test_proba'])
# plot the ROC curve
plt.plot(fpr, tpr)
# plot a secondary diagonal line, with dashed line style and black color to represent a no-skill classifier
plt.plot(fpr, fpr, linestyle = '--', color = 'k')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve');

# Calculate the Area Under the Receiver Operating Characteristic Curve (AUROC) on our test set
AUROC = roc_auc_score(y_test_proba['y_test_class_actual'], y_test_proba['y_hat_test_proba'])
# calculate Gini from AUROC
Gini = AUROC * 2 - 1
# print AUROC and Gini
print('AUROC: %.4f' % (AUROC))
print('Gini: %.4f' % (Gini))

# draw a PR curve
# calculate the no skill line as the proportion of the positive class
no_skill = len(y_test[y_test == 1]) / len(y)
# plot the no skill precision-recall curve
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
# get the values required to plot a PR curve
precision, recall, thresholds = precision_recall_curve(y_test_proba['y_test_class_actual'],
                                                       y_test_proba['y_hat_test_proba'])
# plot PR curve
plt.plot(recall, precision, marker='.', label='Logistic')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
plt.title('PR curve');
```

Image 6: prediction model

<u>Scorecard Development</u>

To finish, a simple scorecard will be used in order to implement the credit risk prediction model. This will allow any person to calculate a client's credit risk score when given certain information. Check the rest of the code in the Jypiter notebook.



```
 create a new dataframe with one column with values from the 'reference_categories' list
f_ref_categories = pd.DataFrame(ref_categories, columns = ['Feature name'])
 We create a second column, called 'Coefficients', which contains only 0 values.
f_ref_categories['Coefficients'] = 0

 Concatenates two dataframes
f_scorecard = pd.concat([summary_table, df_ref_categories])
 reset the index
f_scorecard.reset_index(inplace = True)

 create a new column, called 'Original feature name', which contains the value of the 'Feature name' column
f_scorecard['Original feature name'] = df_scorecard['Feature name'].str.split(':').str[0]

 Define the min and max threshholds for our scorecard
in_score = 300
ax_score = 850

 calculate the sum of the minimum coefficients of each category within the original feature name
in_sum_coef = df_scorecard.groupby('Original feature name')['Coefficients'].min().sum()
 calculate the sum of the maximum coefficients of each category within the original feature name
ax_sum_coef = df_scorecard.groupby('Original feature name')['Coefficients'].max().sum()
 create a new column that has the imputed calculated Score based scaled from the coefficients
f_scorecard['Score - Calculation'] = df_scorecard['Coefficients'] * (max_score - min_score) / (
    max_sum_coef - min_sum_coef)
 update the calculated score of the Intercept
f_scorecard.loc[0, 'Score - Calculation'] = (
    (df_scorecard.loc[0,'Coefficients'] - min_sum_coef) /
    (max_sum_coef - min_sum_coef
    )) * (max_score - min_score) + min_score
 round the values of the 'Score - Calculation' column and store them in a new column
f_scorecard['Score - Preliminary'] = df_scorecard['Score - Calculation'].round()

 check the min and max possible scores of our scorecard
in_sum_score_prel = df_scorecard.groupby('Original feature name')['Score - Preliminary'].min().sum()
ax_sum_score_prel = df_scorecard.groupby('Original feature name')['Score - Preliminary'].max().sum()
rint(min_sum_score_prel)
rint(max_sum_score_prel)

 so both our min and max scores are out by +1. we need to manually adjust this
 Which one? We'll evaluate based on the rounding differences of the minimum category within each Original Feature Name.
d.options.display.max_rows = 102
f_scorecard['Difference'] = df_scorecard['Score - Preliminary'] - df_scorecard['Score - Calculation']

 look like we can get by deducting 1 from the Intercept
f_scorecard['Score - Final'] = df_scorecard['Score - Preliminary']
f_scorecard.loc[0, 'Score - Final'] = 598
```

Image 7: scorecard

Conclusions

In the previous pages, I have given a brief overview of how a credit risk model is built using ML tools with python. These kinds of models are used by banks and other financial institutions to predict what type of customer is behind every loan application. On theoretical basis, credit risk models stem from probability theory in order to deal with uncertainty and stochasticity. Despite of this, methodologically speaking, there are several ways to build a credit risk model. Namely, the expert can use different ML instruments that suits their ultimate goals of a particular model depending on the different features and dimensions present in her data, the type of industry the model is covering (i.e., portfolio management, credit risk management, debt restructuring, NPLs…), the algorithms needed, the type of problem definition…

Apart from the technical implications required to explain credit risk models, my ultimate goal is to display an overall picture on data management using ML tools. Regardless of the small variances of every model, all of them follow a basic structure that consists of data exploration, stratification and feature selection, model training and predictions. Thus, this is a blueprint that one needs to keep in mind when handling large amount of data with a wide variety of features and, if it is the case, labels. In addition to this, a sound mathematical approach it is essential to develop an accurate model. This is mostly, because every case has different problem parameters that are calculated in different ways. Namely, the formula "one mathematical model fits all cases", does not work. This is mainly because of: the type of problem we are solving (classification, regression, clustering…); the type of loss function the expert must choose to get the best predictor; the type of distribution technique needed to select the most relevant features; withing others… Also, the type of data plays a big role in the elaboration of any kind of model using ML techniques, due to "junk data leads to elaborate a junk model" (Elmusrati, 2021).

In short, elaborating a credit risk model (and any kind of other models) with ML tools, is a combination of several factors such as: good data, a well-founded mathematical approach, a concise and clear written code, a reasonable data splitting and model training (to avoid underfitting and overfitting)… With this mix of factors, the expert seeks to construct a model capable to generalize the performance when facing new datapoints not present in the previously. That is, that the expert and the system must reach an efficient process capable of dealing with the uncertainty that dwells in those new inputs. In that way better decisions can be taken in order to reduce the risk for both, the customer and the lender.

References

- Alpaydin, Ethem (2014), Introduction to Machine Learning. 3rd edition. MIT press; Cambridge, Massachusetts, London, England. ISBN 978-0-262-02818-9 (hardcover : alk. paper)

- Data set extracted from Lending Club. Available in Kaggel: https://drive.google.com/file/d/1xaF743cmUgI5kc76I86AeZDE84SMkMnt/view.

- Elmusrati, M. (2021). Lecture on: Probability Theory, Stochastic Processes and Bayesian Decision Theory. ICAT3120. University of Vaasa.

- Goodfellow, I., Bengio, Y., Courville, A. (2016) Deep Learning. MIT Press. Cambridge, Massachusetts, London, England. ISBN: 9780262035613

- Mumtaz, Asad (Aug. 13th 2020a). How to Develop a Credit Risk Model and Scorecard. Towards Data Science. https://towardsdatascience.com/how-to-develop-a-credit-risk-model-and-scorecard-91335fc01f03

- Mumtaz, Asad (Jul. 16th 2020b). How to Find the Best Predictors for ML Algorithms. Toward Data Science. https://towardsdatascience.com/how-to-find-the-best-predictors-for-ml-algorithms-4b28a71a8a80

- Müller, Andreas C. & Guido, S. (2017) Introduction to Machine Learning with Python. A Guide for Data Scientists. O'reilly. California, USA. ISBN: 978-1-449-36941-S

- Rogers, S & Girolami, M. 2012. Afirst Course in Machine Learning. 2nd edition. Taylor and Francis. Cambridge, United Kingdom. ISBN: 978-1-4398-2414-6

- Russell, S & Norvig, P. (2010). Artificial Intelligence. A modern approach. 3rd Edition. Pearson Education Inc. New Jersey. ISBN-13: 978-0-13-604259-4

- Yvan Saeys, Iñaki Inza, Pedro Larrañaga, A review of feature selection techniques in bioinformatics, *Bioinformatics*, Volume 23, Issue 19, 1 October 2007, Pages 2507–2517, https://doi.org/10.1093/bioinformatics/btm344