Modelos predictivos y la relevancia del Machine Learning

Enunciado:

Las metodologías estudiadas han resultado bastante valiosas para el equipo de analistas y han construido algunos modelos que responden preguntas interesantes de negocio. Sin embargo, hay una preocupación porque los datos que utilizan son bastante cambiantes lo que hace que los modelos puedan descalibrarse rápidamente. Buscando que las herramientas analíticas puedan tener continuidad en el tiempo y que respondan a preguntas del negocio de manera continua, se ha incorporado, como parte del estudio de metodologías, el Machine learning y los modelos supervisados y no supervisados.

Objetivos:

- Hacer un comparativo de metodologías estadísticas para análisis de grandes volúmenes de datos y construir un árbol de recomendación sobre las más adecuadas para la empresa.
- Hacer un ejercicio de entendimiento de los modelos utilizados para el análisis de riesgos financieros.
- Construir un modelo de scoring de riesgos.

La Figura 1 contiene un árbol de recomendación elaborado por sklearn el cual contiene diferentes modelos de Machine Learning dependiendo de la cantidad de datos y el resultado buscado.

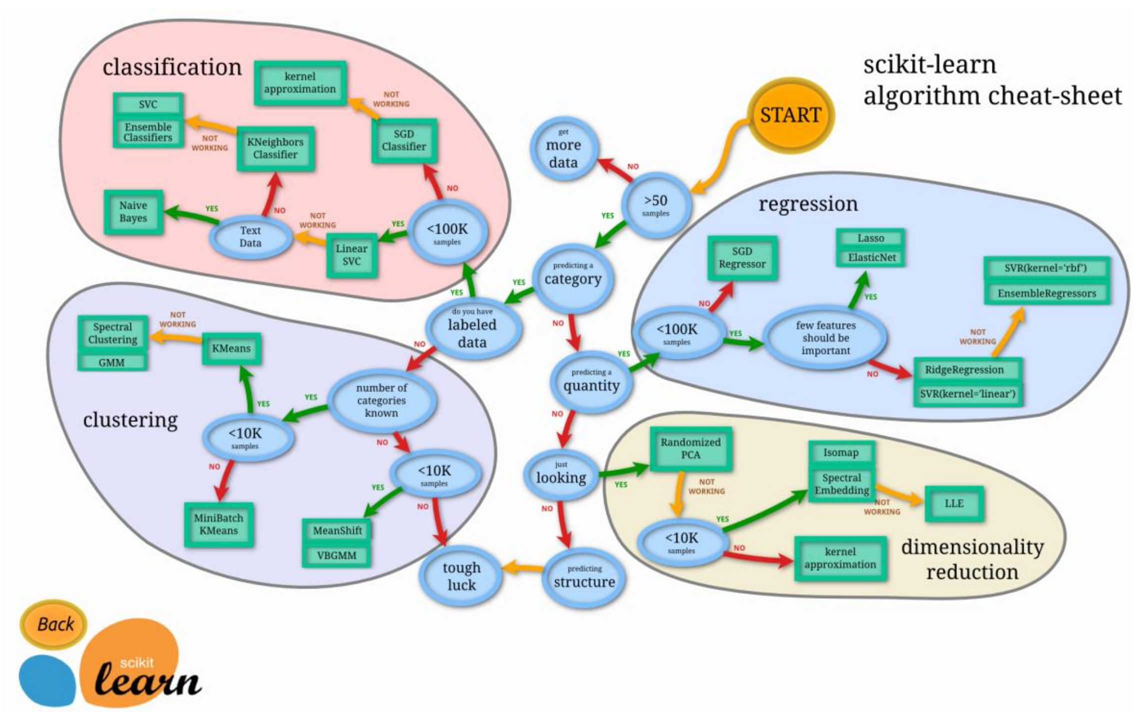

Fig 1. Árbol de modelos de Sklearn

In [53]:
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR, SVC
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import accuracy_score, classification_report
```

In [4]:
```python
df = pd.read_csv("./creditInfo.csv")
```

1. Dataset analysis

Tabla 1. Diccionario de datos de la fuente creditInfo

Feature Name	Description
person_age	Age
person_income	Anual Income
personhomeownership	Home Ownership
personemplength	Employment Length (in years)
loan_intent	Loan intent
loan_grade	Loan grade
loan_amnt	Loan amount
loanintrate	Interest rate
loan_status	Loan status (0 is non default 1 is default)
loanpercentincome	Percent Income
cbpersondefaultonfile	Historical default
cbpersoncredhistlength	Credit History Length

Analyzing the loan amount intervals given depending on the credit's grade

In [30]:
```python
grades = []
for grade in set(df["loan_grade"]): #set para obtener valores unicos
    #dataframe para analizar rangos de precios
    grades.append([
        grade, #nombre de columna
        df[df["loan_grade"] == grade]["loan_amnt"].min(), #precio minimo
        df[df["loan_grade"] == grade]["loan_amnt"].max() #precio maximo
    ])
grades = pd.DataFrame(data  = grades, columns = ["Grade", "MinLoanAmount", "MaxLoanAmount"]).sort_values("Mi
grades
```

Out[30]:

	Grade	MinLoanAmount	MaxLoanAmount
1	G	1600	35000
2	F	1200	35000
0	E	1000	35000
5	D	1000	35000
3	C	500	35000
4	A	500	35000
6	B	500	35000

Analyzing the credit's grade variation acording to the person's credit history

In [47]:
```python
credits = []
for credit in set(df["cb_person_cred_hist_length"]):
    credits.append([
        credit, #cantidad de creditos de la persona
        list(sorted(set(df["loan_grade"][df["cb_person_cred_hist_length"] == credit]))) #lista de calificaci
    ])
credits = pd.DataFrame(data = credits,columns = ["CreditsNum", "Grades"])
credits
```

Out[47]:

	CreditsNum	Grades
0	2	[A, B, C, D, E, F, G]
1	3	[A, B, C, D, E, F, G]
2	4	[A, B, C, D, E, F, G]
3	5	[A, B, C, D, E, F, G]
4	6	[A, B, C, D, E, F, G]
5	7	[A, B, C, D, E, F, G]
6	8	[A, B, C, D, E, F, G]
7	9	[A, B, C, D, E, F, G]
8	10	[A, B, C, D, E, F, G]
9	11	[A, B, C, D, E, F, G]
10	12	[A, B, C, D, E, F, G]
11	13	[A, B, C, D, E, F, G]
12	14	[A, B, C, D, E, F, G]
13	15	[A, B, C, D, E, F, G]
14	16	[A, B, C, D, E, F]
15	17	[A, B, C, D, E, F]
16	18	[A, B, C, D, E]
17	19	[A, B, C, D, F]
18	20	[A, B, C, D, F]
19	21	[A, B, C, D]
20	22	[A, B, C, D, E]
21	23	[A, B, C, D]
22	24	[A, B, C, D, E]
23	25	[A, B, C, D, E]
24	26	[A, B, C, D]
25	27	[A, B, C, D]
26	28	[A, B, C, D, E]
27	29	[A, B, C, D, E]
28	30	[A, B, C, D, E]

Analyzing data correlations

In [53]:
```python
df.corr(method="pearson")
```

Out[53]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_i
person_age	1.000000	0.173202	0.163106	0.050787	0.012580	-0.021629	-0.
person_income	0.173202	1.000000	0.134268	0.266820	0.000792	-0.144449	-0.
person_emp_length	0.163106	0.134268	1.000000	0.113082	-0.056405	-0.082489	-0.
loan_amnt	0.050787	0.266820	0.113082	1.000000	0.146813	0.105376	0.
loan_int_rate	0.012580	0.000792	-0.056405	0.146813	1.000000	0.335133	0.
loan_status	-0.021629	-0.144449	-0.082489	0.105376	0.335133	1.000000	0.
loan_percent_income	-0.042411	-0.254471	-0.054111	0.572612	0.120314	0.379366	1.
cb_person_cred_hist_length	0.859133	0.117987	0.144699	0.041967	0.016696	-0.015529	-0.

Analyzing type Object values

In [60]:
```python
df.select_dtypes(object)
```

Out[60]:

	person_home_ownership	loan_intent	loan_grade	cb_person_default_on_file
0	RENT	PERSONAL	D	Y
1	OWN	EDUCATION	B	N
2	MORTGAGE	MEDICAL	C	N
3	RENT	MEDICAL	C	N
4	RENT	MEDICAL	C	Y
...	...	...	...	...
32576	MORTGAGE	PERSONAL	C	N
32577	MORTGAGE	PERSONAL	A	N
32578	RENT	HOMEIMPROVEMENT	B	N
32579	MORTGAGE	PERSONAL	B	N
32580	RENT	MEDICAL	B	N

32581 rows × 4 columns

In [59]:
```python
#seeing data imabalances
df["loan_grade"].value_counts()
```

Out[59]:
```
A    9402
B    9151
C    5699
D    3248
E     870
F     209
G      59
Name: loan_grade, dtype: int64
```

2. Data Preprocessing

In [5]:
```python
cat_values = df.select_dtypes(object) #selecting all categorical values
```

In [6]:
```python
le = LabelEncoder()
num_cat_values = pd.DataFrame() #dataframe for storing the numerical value obtained from the categorical var
for column in cat_values.columns:
    num_cat_values[column] = le.fit_transform(cat_values[column])
num_cat_values.sample(3)
```

Out[6]:

	person_home_ownership	loan_intent	loan_grade	cb_person_default_on_file
2692	3	5	1	0
14800	0	2	0	0
19480	0	4	0	0

In [7]:
```python
#Getting an all numerical values dataframe
num_data = pd.concat([df.select_dtypes(np.number), num_cat_values], axis = 1)
num_data.sample(3)
```

Out[7]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_c
27919	30	21600	1.0	1500	11.71	1	0.07	
7541	22	53808	5.0	6000	8.90	0	0.11	
21387	29	48000	0.0	6000	14.27	0	0.13	

3. Further analysis

In [72]:
```python
#Analyzing data correlations
num_data.corr(method = "pearson")
```

Out[72]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_i
person_age	1.000000	0.173202	0.163106	0.050787	0.012580	-0.021629	-0.
person_income	0.173202	1.000000	0.134268	0.266820	0.000792	-0.144449	-0.
person_emp_length	0.163106	0.134268	1.000000	0.113082	-0.056405	-0.082489	-0.
loan_amnt	0.050787	0.266820	0.113082	1.000000	0.146813	0.105376	0.
loan_int_rate	0.012580	0.000792	-0.056405	0.146813	1.000000	0.335133	0.
loan_status	-0.021629	-0.144449	-0.082489	0.105376	0.335133	1.000000	0.
loan_percent_income	-0.042411	-0.254471	-0.054111	0.572612	0.120314	0.379366	1.
cb_person_cred_hist_length	0.859133	0.117987	0.144699	0.041967	0.016696	-0.015529	-0.
person_home_ownership	-0.032506	-0.203177	-0.231736	-0.130776	0.140454	0.211600	0.
loan_intent	0.035518	0.001527	0.021749	-0.004597	-0.001357	-0.065575	0.
loan_grade	0.014218	-0.001022	-0.047276	0.145799	0.933684	0.373080	0.
cb_person_default_on_file	0.005807	-0.003613	-0.027728	0.039081	0.501072	0.179141	0.

4. Model selection

Acording to Rudra and Kumar a first approach wa taken in 2016 by using genetic programming and deep learning techniques. Further in time, explicitly in 2018 Random Forest were studied to see their prefomance in this specific problem [1, pp 5 - 7].

For their part Ossa and Jaramillo stated that logistic regression, random forests and SVM's were used aswell for this purpose [2, pp 9 - 11].

4.1 Regression

4.1.1 SVR

In [18]:
```python
#droping NAN rows
num_data = num_data.dropna()
```

In [19]:
```python
#normalizing the data
scaler = StandardScaler()
normalized_data = scaler.fit_transform(num_data)
normalized = pd.DataFrame(normalized_data, columns = num_data.columns.tolist())
```

In [20]:
```python
#splitting the data 80% training 20% testing
X = normalized.drop(columns = ["loan_grade"])
Y = normalized["loan_grade"]
x_train, x_test, y_train, y_test = train_test_split(X,Y,train_size=0.8,random_state = 42)
```

In [23]:
```python
#training and testing SVR
svr = SVR()
svr.fit(x_train, y_train)
print(
    f"The models accuracy in training is: {svr.score(x_train, y_train)}"\
    f"\nThe models accuracy in testing is: {svr.score(x_test, y_test)}"
    )
```

```
The models accuracy in training is: 0.9226277987395867
The models accuracy in testing is: 0.9132143220735378
```

4.1.2 Random Forest

With normalized data

In [27]:
```python
rfrn = RandomForestRegressor(n_estimators = 21, max_depth= 7) #21 trees, 7 max nodes
rfrn.fit(x_test, y_test)
```

Out[27]:
```
RandomForestRegressor
RandomForestRegressor(max_depth=7, n_estimators=21)
```

Without normalized data

In [28]:
```python
rfr = RandomForestRegressor(n_estimators = 21, max_depth= 7) #21 trees, 7 max nodes
rfr.fit(x_test, y_test)
```

Out[28]:
```
RandomForestRegressor
RandomForestRegressor(max_depth=7, n_estimators=21)
```

In [30]:
```python
print(
    f"Random Forest Regression with normalization score on training: {rfrn.score(x_train, y_train)}"\
    f"\nRandom Forest Regression with normalization score on testing: {rfrn.score(x_test, y_test)}"\
    f"\nRandom Forest Regression without normalization score on training: {rfr.score(x_train, y_train)}"\
    f"\nRandom Forest Regression without normalization score on testing: {rfr.score(x_test, y_test)}"
)
```

```
Random Forest Regression with normalization score on training: 0.9615219179295987
Random Forest Regression with normalization score on testing: 0.9735735138767163
Random Forest Regression without normalization score on training: 0.9618576407709001
Random Forest Regression without normalization score on testing: 0.9735558946145125
```

4.2 Classification

4.2.1 SVM

In [37]:
```python
#dropping NAN rows
num_data = num_data.dropna()
```

Without normalized data

In [40]:
```python
#splitting into training and testing data
X = num_data.drop(columns= "loan_grade")
Y = num_data["loan_grade"]
x_train, x_test, y_train, y_test = train_test_split(X, Y,random_state=42, train_size=0.8)
```

In [41]:
```python
#training the model
svc = SVC()
svc.fit(x_train, y_train)
predict = svc.predict(x_train)
```

In [44]:
```python
print(
    f"SVM without normalization score on training: {accuracy_score(y_train, predict)}"
    f"\nSVM without normalization score on testing: {accuracy_score(y_test, svc.predict(x_test))}"
)
```

```
SVM without normalization score on training: 0.36817983413356614
SVM without normalization score on testing: 0.36557262569832405
```

With normalized data

In [45]:
```python
scaler = StandardScaler()
normalized_data = scaler.fit_transform(num_data)
normalized = pd.DataFrame(normalized_data, columns = num_data.columns.tolist())
```

In [49]:
```python
X = normalized.drop(columns= "loan_grade")
Y = df["loan_grade"]
x_train, x_test, y_train, y_test = train_test_split(X, Y,random_state=42, train_size=0.8)
```

In [56]:
```python
#training the model
svc = SVC()
svc.fit(x_train, y_train)
train_predict = svc.predict(x_train)
test_predict = svc.predict(x_test)
```

In [57]:
```python
print(
    f"SVM with normalization score on training: {accuracy_score(y_train, train_predict)}"
    f"\nSVM with normalization score on testing: {accuracy_score(y_test, test_predict)}"
)
```

```
SVM with normalization score on training: 0.8969009166302925
SVM with normalization score on testing: 0.884427374301676
```

In [58]:
```python
print(classification_report(y_test, svc.predict(x_test)))
```

```
              precision    recall  f1-score   support

           A       0.98      0.97      0.97      1893
           B       0.92      0.96      0.94      1830
           C       0.84      0.84      0.84      1102
           D       0.67      0.76      0.71       657
           E       0.50      0.26      0.34       195
           F       0.67      0.05      0.09        43
           G       1.00      0.38      0.55         8

    accuracy                           0.88      5728
   macro avg       0.80      0.60      0.63      5728
weighted avg       0.88      0.88      0.88      5728
```

4.2.2 Random Forest

In [60]:
```python
#dropping NAN rows
num_data = num_data.dropna()
```

In [66]:
```python
#splitting into training and testing data
X = num_data.drop(columns= "loan_grade")
Y = df["loan_grade"]
x_train, x_test, y_train, y_test = train_test_split(X, Y,random_state=42, train_size=0.8)
```

In [67]:
```python
rfc = RandomForestClassifier(n_estimators = 21, max_depth = 7)
rfc.fit(x_train, y_train)
```

Out[67]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=7, n_estimators=21)
```

In [68]:
```python
train_predict = rfc.predict(x_train)
test_predict =  rfc.predict(x_test)
print(
    f"Random Forest Classification score on training: {accuracy_score(y_train, train_predict)}"\
    f"\nRandom Forest Classification score on testing: {accuracy_score(y_test, test_predict)}"
)
```

```
Random Forest Classification score on training: 0.8972064600611087
Random Forest Classification score on testing: 0.8889664804469274
```

In [69]:
```python
print(classification_report(y_test, test_predict))
```

```
              precision    recall  f1-score   support

           A       0.98      0.98      0.98      1893
           B       0.92      0.98      0.95      1830
           C       0.88      0.84      0.86      1102
           D       0.63      0.79      0.70       657
           E       0.54      0.04      0.07       195
           F       0.00      0.00      0.00        43
           G       1.00      0.12      0.22         8

    accuracy                           0.89      5728
   macro avg       0.71      0.53      0.54      5728
weighted avg       0.88      0.89      0.87      5728
```

```
c:\Users\Admin\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\metrics\_classification.py:1
334: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no pr
edicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Admin\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\metrics\_classification.py:1
334: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no pr
edicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\Admin\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\metrics\_classification.py:1
334: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no pr
edicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

5. References

[1] M. Rudra Kumar, V. Kumar Gunjan, "Review of Machine Learning models for Credit Scoring Analysis," Revista Ingeniería Solidaria, vol. 16, no. 1, 2020. doi: https://doi.org/10.16925/2357-6014.2020.01.11

[2] Ossa Giraldo, W. y Jaramillo Marín, V. (2021). Machine Learning para la estimación del riesgo de crédito en una cartera de consumo. Repositorio Institucional Universidad EAFIT.
https://repository.eafit.edu.co/bitstream/handle/10784/29589/Wbeimar_OssaGiraldo_Veronica_JaramilloMarin_2021.pdf?sequence=2&isAllowed=y