POLITECNICO
MILANO 1863

NAML Project: credit card fraud detection

Karanbir Singh
Matteo Vitali

Academic year 2024-2025

The paper

Ileberi *et al. Journal of Big Data* (2022) 9:24
https://doi.org/10.1186/s40537-022-00573-8

Journal of Big Data

RESEARCH Open Access

A machine learning based credit card fraud detection using the GA algorithm for feature selection

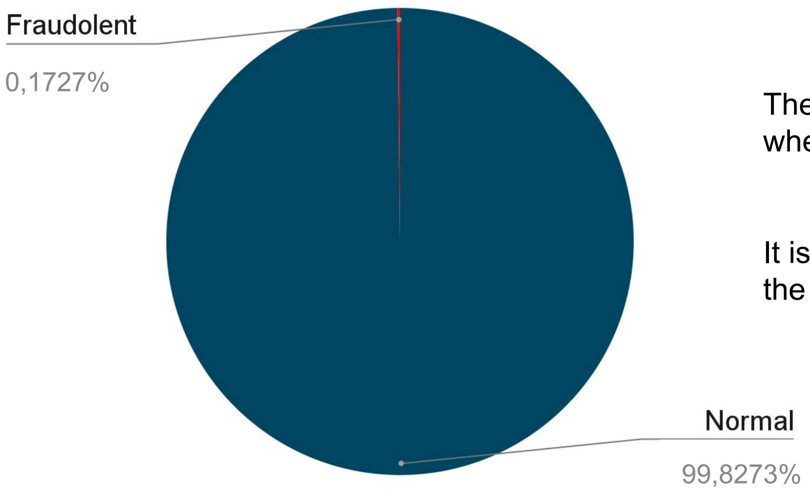
Emmanuel Ileberi[1*], Yanxia Sun[1] and Zenghui Wang[2]

SpringerOpen

Dataset

Features

Time	V1	V2	…	…	V28	Amount	Class



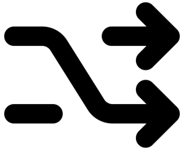
The dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions.

It is *highly unbalanced*:
the positive class (frauds) account for 0.1727% of all transactions.

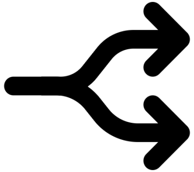
Preprocessing

GA features selection

Shuffle + Split



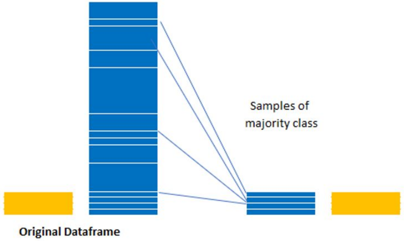
+



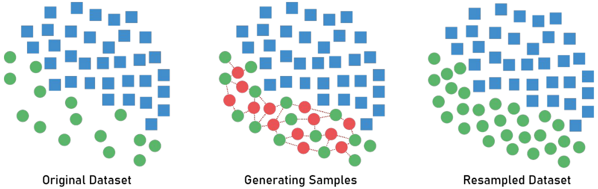
Normalization

$$f_s = \frac{f - min(f)}{max(f) - min(f)}$$

Undersampling



Synthetic Minority Oversampling Technique



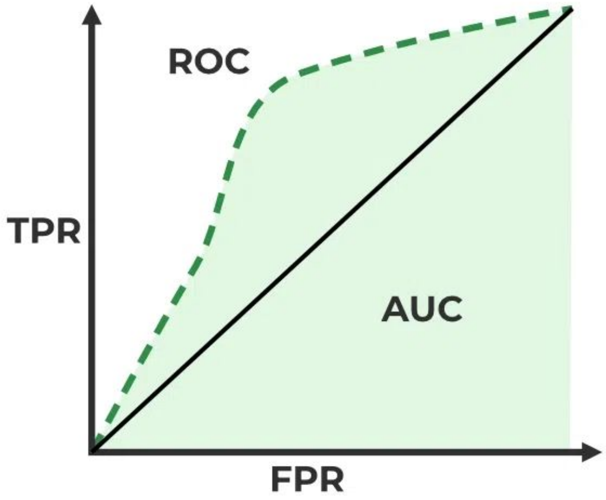
Metrics

Accuracy $$AC = \frac{TN + TP}{TP + TN + FP + FN}$$

Recall $$RC = \frac{TP}{FN + TP}$$

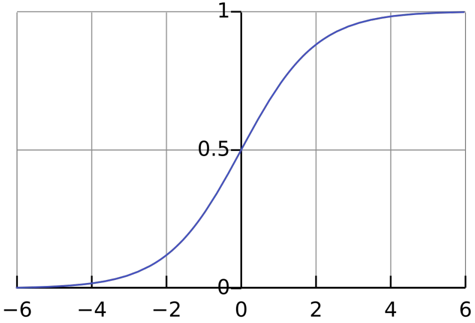
Precision $$PR = \frac{TP}{FP + TP}$$

F1-score $$F1 = 2 \cdot \frac{PR \cdot RC}{PR + RC}$$



Logistic Regression

Based on *discriminative approach*, it models directly the conditioned class probability:



C_1 : fraud transaction

C_2 : normal transaction

Model definition

$$z = w_0 + \sum_{i=1}^{n} w_i X_i = w_0 + w_1 X_1 + w_2 X_2 + \ldots + w_n X_n$$

$$p(C_1|z) = \hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$$

$$predicted\_label = \begin{cases} 1 & \text{if } \hat{y} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

$$p(C_2|z) = 1 - p(C_1|z)$$

Logistic Regression

Optimization algorithm - RMSProp

$$E[g^2]^{(k)} = \gamma E[g^2]^{(k-1)} + (1-\gamma) g^{2(k)}$$

$$\omega^{(k+1)} = \omega^{(k)} - \frac{\eta}{\sqrt{E[g^2] + \epsilon}} g^{(k)}$$

Loss function

$$J_{x-entropy} = -\frac{1}{N} \sum_{i}^{N} y_i \log(\hat{y}_i) + (1-y_i) \log(1-\hat{y}_i)$$

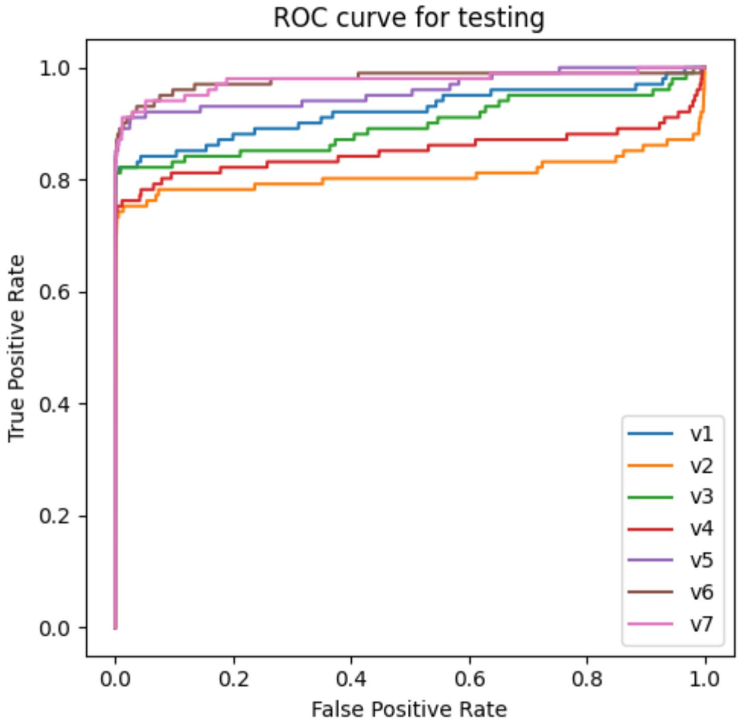
Different tunable parameters to maximize the quality of the model and prevent overfitting:

epochs ***batch_size*** ***learning_rate*** ***decay_rate*** ***(regularization): penalization***

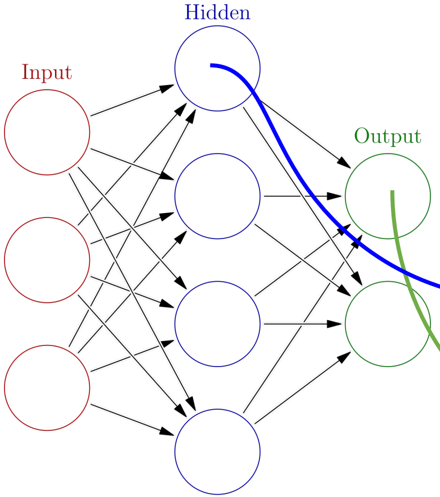
Low memory**, **high number of iterations and batch size during training and hard to find a balance for improving all metrics

Results of LR

Feature vector	Accuracy	Recall	Precision	F1-Score	AUC
v_1	99.91	56.43	85.07	67.86	0.92
	99.91	57.52	82.27	67.70	
v_2	99.88	35.64	87.80	50.70	0.81
	99.89	47.78	79.41	59.66	
v_3	99.90	54.46	85.94	66.67	0.90
	99.90	53.09	80.00	63.82	
v_4	99.87	30.69	86.11	45.26	0.85
	99.89	46.90	77.94	58.56	
v_5	99.92	63.37	87.67	73.56	0.96
	99.77	46.70	34.64	39.84	
v_6	99.93	68.32	89.61	77.53	0.98
	93.88	60.17	62.96	61.53	
v_7	99.93	67.33	90.67	77.27	0.98
	79.91	59.29	81.70	68.71	



Neural network schema



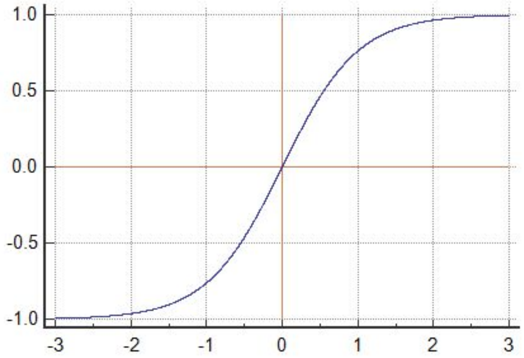
Model definition

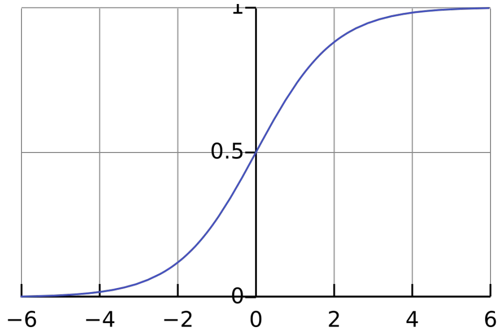
$$\mathbf{x}^{(l)} = \sigma(\mathbf{z}^{(l)})$$
$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}$$

Activation functions

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$





Artificial Neural Network

Optimization algorithm - RMSProp

$$E[g^2]^{(k)} = \gamma E[g^2]^{(k-1)} + (1-\gamma) g^{2(k)}$$

$$\omega^{(k+1)} = \omega^{(k)} - \frac{\eta}{\sqrt{E[g^2] + \epsilon}} g^{(k)}$$

Loss function

$$J_{x-entropy} = -\frac{1}{N} \sum_{i}^{N} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Different tunable parameters to maximize the quality of the model and prevent overfitting:

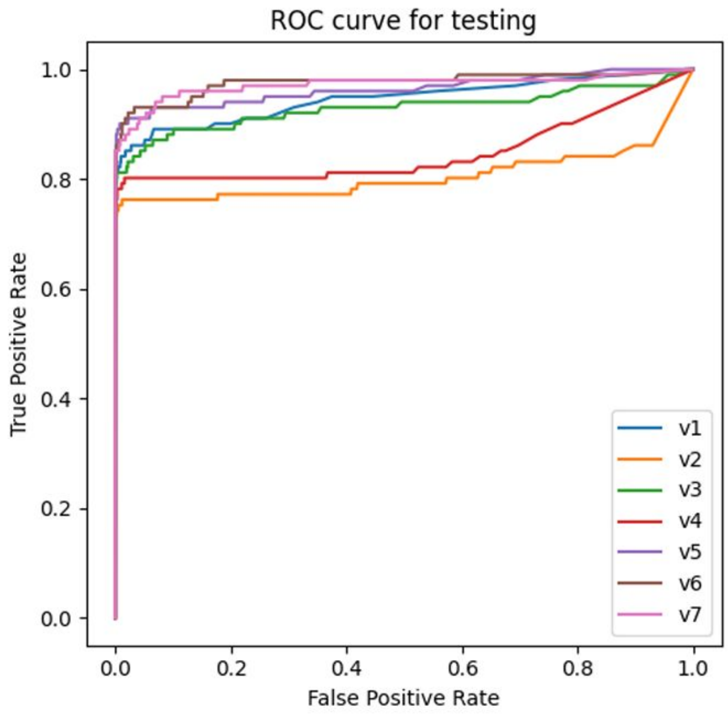
act_func *out_act_func* *epochs* *batch_size* *learning_rate* *decay_rate*

(regularization): penalization

Good performances with low number of iterations and batch size (w.r.t LR),
a small change of hyperparameters results in completely different performances

Results of ANN

Feature vector	Accuracy	Recall	Precision	F1-Score	AUC
v_1	99.92	67.33	83.95	74.73	0.95
	99.94	77.87	84.61	81.10	
v_2	99.91	66.34	79.76	72.43	0.81
	99.91	66.37	76.53	71.09	
v_3	99.92	76.24	80.21	78.17	0.93
	99.91	67.25	77.55	72.03	
v_4	99.92	75.25	80.85	77.95	0.85
	99.91	61.06	81.17	69.69	
v_5	99.92	83.17	75.68	79.25	0.96
	99.08	77.87	12.27	21.20	
v_6	99.93	81.19	79.61	80.39	0.98
	97.80	74.33	42.85	54.36	
v_7	99.93	82.18	80.58	81.37	0.97
	88.93	78.76	82.40	80.54	



Decision Trees

Different *impurity* measures supported by the algorithm, Gini and entropy:

$$\mathcal{G}(y) = 1 - \sum_{k=1}^{K} p_k^2 \qquad \mathcal{H}(y) = -\sum_{k=1}^{K} p_k \log_2(p_k)$$

Non-parametric model that seeks to best fit the data (i.e. *information gain*):

$$\mathcal{I}_G(y) = \mathcal{I}(y) - \text{weighted impurity of children} = \mathcal{I}(y) - \left(\frac{n_{left}}{n} \cdot \mathcal{I}(y_{left}) + \frac{n_{right}}{n} \cdot \mathcal{I}(y_{right})\right)$$

Different tunable parameters to prevent overfitting:

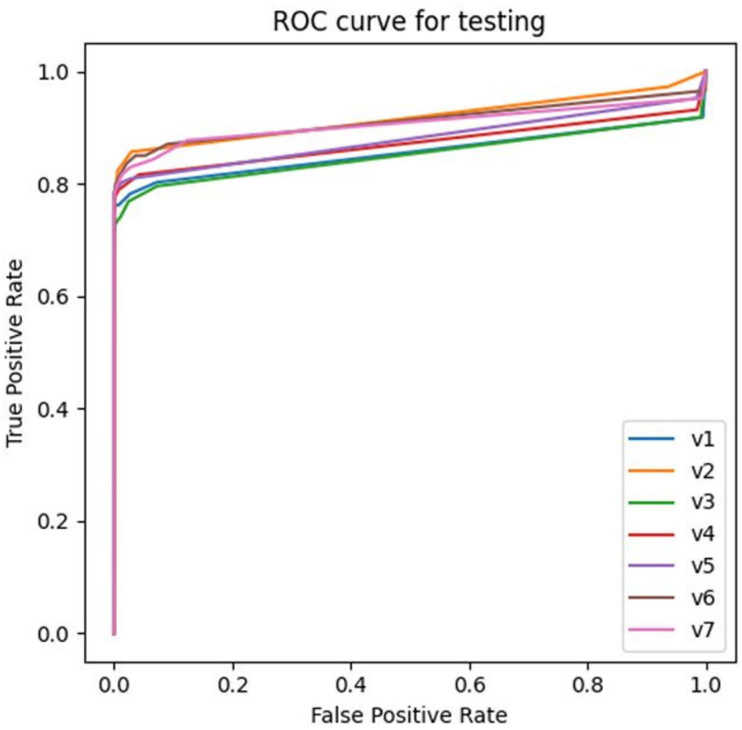
max_depth ***min_samples_split*** ***min_samples_leaf*** ***min_impurity_decrease*** ***max_features***

The tree is built in a DFS-like manner until one stopping criteria is met.

Little time and memory to train**,** ***risk of heavy overfitting*****.**

Results of DT

Feature vector	Accuracy	Recall	Precision	F1-Score	AUC
v_1	99.94	75.51	87.40	81.02	0.86
	99.92	75.22	75.22	75.22	
v_2	99.93	73.46	85.04	78.83	0.92
	99.87	68.14	60.62	64.16	
v_3	99.93	68.03	87.72	76.63	0.86
	99.90	76.10	68.80	72.26	
v_4	99.94	73.47	87.10	79.70	0.87
	99.91	76.10	72.26	74.13	
v_5	99.94	76.19	91.80	83.27	0.88
	99.89	72.56	65.07	68.61	
v_6	99.94	74.83	89.43	81.48	0.91
	96.91	76.10	71.07	73.50	
v_7	99.94	74.15	87.90	80.44	0.91
	89.91	79.64	68.70	73.77	



Random Forest

Build some *bootstrapped datasets* considering every time a subset of features.

Train a DT for each of them.

Make predictions by *majority voting*:

$$\tilde{y} = \arg\max_k \sum_{i=1}^{B} \mathbb{I}\{T_i(x) = k\}$$

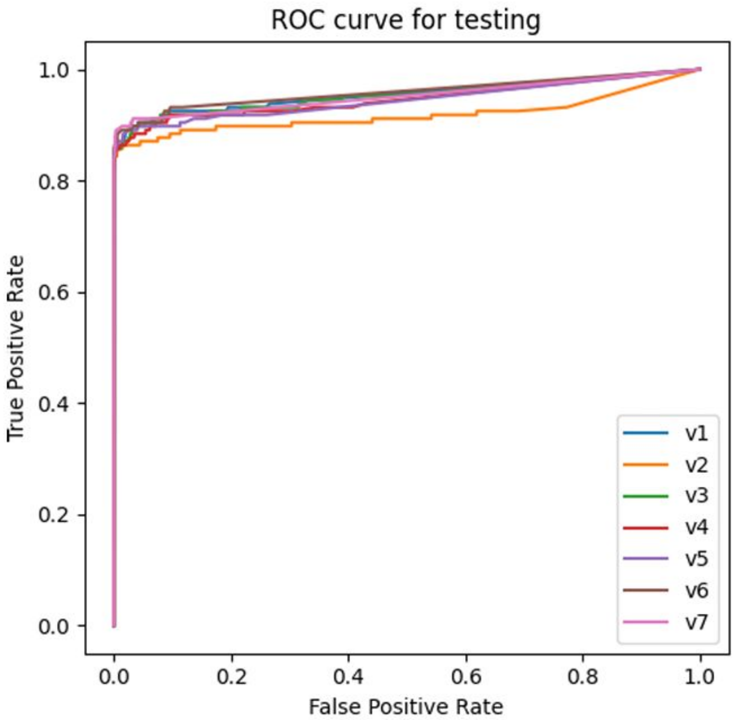
Or by *probability averaging*:

$$\tilde{y} = \arg\max_k \frac{1}{B} \sum_{i=1}^{B} P\{k \text{ assigned to } x \text{ by tree } T_i\}$$

***More time and memory to train*, *but better results than DT since overfitting is mitigated*.**

Results of RF

Feature vector	Accuracy	Recall	Precision	F1-Score	AUC
v_1	99.95	80.95	91.53	85.92	0.96
	99.94	76.99	89.69	82.85	
v_2	99.94	78.23	81.08	87.12	0.92
	99.93	76.10	82.69	79.26	
v_3	99.94	79.59	87.97	83.57	0.95
	99.94	75.22	85.85	80.18	
v_4	99.94	80.27	88.06	83.99	0.95
	99.94	77.87	83.80	80.73	
v_5	99.96	80.95	92.25	86.23	0.94
	99.98	72.56	95.34	82.41	
v_6	99.95	82.31	89.63	85.82	0.96
	87.95	77.87	92.63	84.61	
v_7	99.95	80.95	88.81	84.70	0.95
	83.78	79.64	92.78	85.71	



Naive Bayes

Based on *conditional independence* assumption of features given the label:

$$P\{X = x_i | Y = y_j, Z = z_k\} = P\{X = x_i | Z = z_k\} \quad \forall i, j, k$$

And *Bayes rule*:

$$P\{Y = y_i | X = x_k\} = \frac{P\{X = x_k | Y = y_i\} \cdot P\{Y = y_i\}}{\sum_j P\{X = x_k | Y = y_j\} \cdot P\{Y = y_j\}}$$

The predicted label is given by:

$$Y = \arg\max_{y_k} \frac{P\{Y = y_k\} \cdot \prod_i P\{X_i | Y = y_k\}}{\sum_j P\{Y = y_j\} \cdot \prod_i P\{X_i | Y = y_i\}}$$

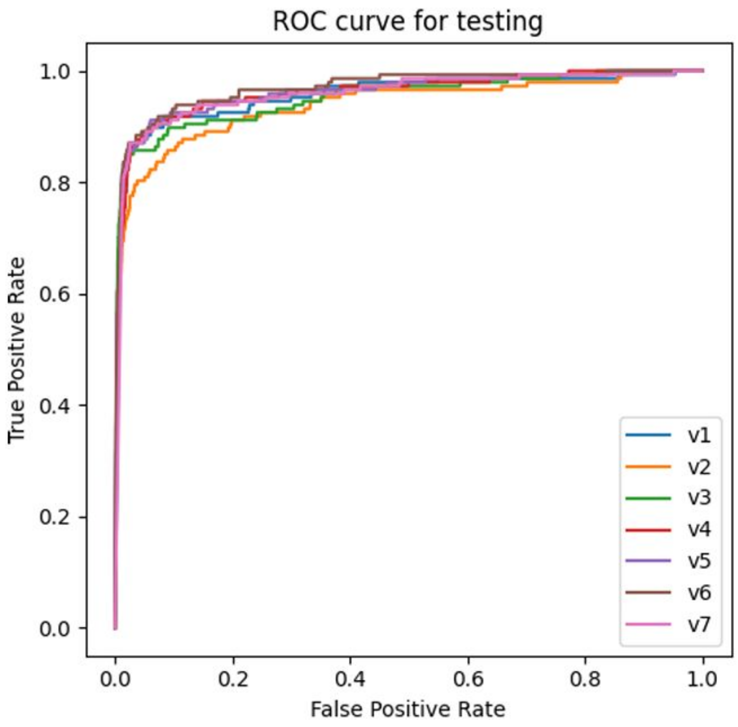
Mean and variance are estimated with *unbiased estimators*:

$$\hat{\mu}_{i,k} = \frac{1}{\sum_j \mathbb{I}\{Y_j = y_k\}} \cdot \sum_j X_{i,j} \cdot \mathbb{I}\{Y_j = y_k\} \qquad \hat{\sigma}^2_{i,k} = \frac{1}{\left(\sum_j \mathbb{I}\{Y_j = y_k\}\right) - 1} \cdot \sum_j (X_{i,j} - \hat{\mu}_{i,k})^2 \cdot \mathbb{I}\{Y_j = y_k\}$$

***Struggles to capture underlying structure (high recall or high precision)*, *but very fast on training and inference*.**

Results of NB

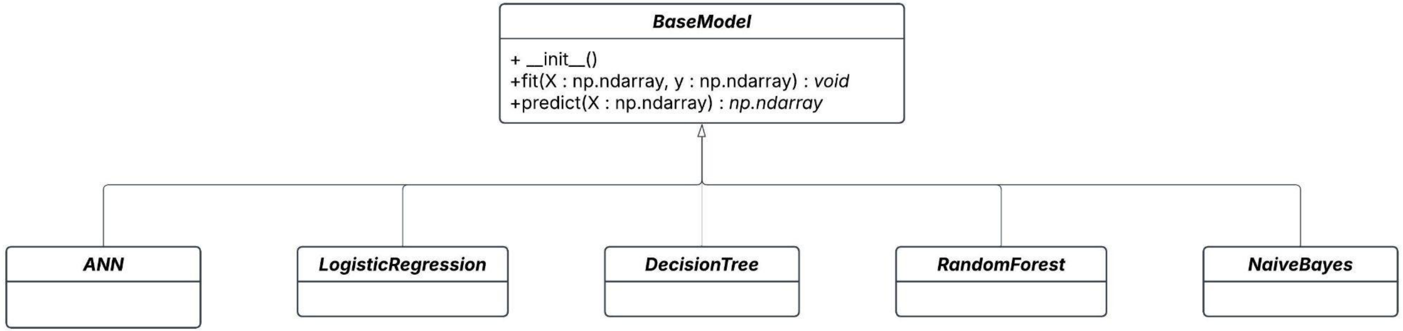
Feature vector	Accuracy	Recall	Precision	F1-Score	AUC
v_1	98.10	83.67	7.15	13.17	0.96
	98.13	84.95	6.83	12.65	
v_2	98.49	70.07	7.65	13.8	0.94
	98.65	77.87	8.59	15.47	
v_3	98.69	80.95	9.85	17.56	0.95
	98.81	81.41	10.07	17.93	
v_4	98.31	75.51	7.29	13.29	0.96
	98.48	81.41	7.97	14.53	
v_5	99.38	58.50	15.61	24.64	0.96
	99.4	57.52	15.85	24.85	
v_6	99.30	64.63	14.82	24.11	0.97
	80.31	64.60	13.95	22.95	
v_7	97.84	83.67	6.32	11.76	0.96
	78.14	83.18	6.73	12.46	



Polymorphism of models

Each model extends a class called *“BaseModel”.*

The GA sees each classifiers as a “*BaseModel”* on which it can call only the exposed methods.



Consequently, required arguments and output types are always the same.

Additional arguments can be passed as default-valued arguments.

Genetic Algorithm for feature selection

Individuals are represented by one chromosome: each gene is 1 if the corresponding feature is selected.

$$c(A) = (b_1, b_2, \dots, b_{|F|}), \quad \text{where } b_i = \begin{cases} 1, & \text{if } f_i \in A, \\ 0, & \text{otherwise.} \end{cases}$$

$$X = \mathcal{P}(F) = \{A \mid A \subseteq F\}$$

$$c^{-1}(b) = \{f_i \in F \mid b_i = 1\}$$

Size of the population is fixed: $\mathcal{B}_t = (b_{1,t}, b_{2,t}, \dots, b_{m,t})$

Selection is based on *roulette wheel* policy:

$$P\{b_{i,t} \text{ is selected}\} = \frac{\phi(b_{i,t})}{\sum_{k=1}^{m} \phi(b_{k,t})}$$

***Finds good feature vectors*, takes a lot for computationally intensive models**

Crossover is available on *single point* and *multiple points.*

Mutation helps avoiding local maxima and is applied with probability p_m.

For fitness we can use a single metric (usually *accuracy)* or a combination of them (e.g. *accuracy* + *precision)*

Results of GA

Model Type	Fitness Function	Feature Vector	Fitness Value
Naive Bayes	`accuracy(x, y)`	$Time$, V_4, V_9, V_{11}, V_{16}, V_{17}, V_{18}, V_{24}, V_{26}, V_{27}	0.9988
Naive Bayes	`2.0 * accuracy(x, y) + precision(x, y)`	V_4, V_5, V_6, V_{10}, V_{14}, V_{15}, V_{16}, V_{17}, V_{18}, V_{21}, V_{22}, V_{26}, V_{28}, $Amount$	2.8357
Random Forest	`accuracy(x, y)`	V_2, V_4, V_6, V_7, V_8, V_9, V_{10}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{18}, V_{19}, V_{21}, V_{23}, V_{25}, V_{27}, $Amount$	0.9996
Decision Tree	`accuracy(x, y)`	$Time$, V_1, V_2, V_4, V_5, V_6, V_7, V_9, V_{10} V_{11}, V_{12}, V_{13}, V_{14}, V_{15}, V_{16}, V_{18}, V_{20}, V_{21}, V_{22}, V_{23}, V_{24}, V_{27}	0.9995
Artificial Neural Network	`accuracy(x, y)`	$Time$, V_1, V_5, V_6, V_7, V_8, V_{10}, V_{11}, V_{13} V_{14}, V_{16}, V_{18}, V_{19}, V_{20}, V_{21}, V_{24}, V_{25}, V_{26}	0.9995
Logistic Regression	`accuracy(x, y)`	V_3, V_4, V_5, V_6, V_8, V_9, V_{10}, V_{11}, V_{12} V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, V_{20}, V_{21}, V_{24}, V_{25}, V_{26}, $Amount$	0.9993

Conclusions

Overall good performances of all the models (except NB).

GA is performing well: feature vector maximizing accuracy in RF aren’t that good for other classifiers.

Authors have used Scikit-Learn, our implementation is from scratch:
not that big gap in performances!

Different metrics for different stakeholders: ***accuracy vs recall***.

Models require very different times for training and inference: ***performance vs time***.

Hyper-parameters and post-training (especially for DT) optimization not implemented:
performances can still improve!