Binary classification for different datasets

Bogdan Chiorean

January 10th, 2025

1 Problem statement

In this work, we investigate binary classification on two different datasets, each with its own set of difficulties and traits.The first dataset includes a wide range of demographic and financial characteristics related to insurance issuing. The second dataset, which has fewer variables, is about credit card application approval, with an emphasis on critical financial indicators that influence decisions.

Due to the disparate characteristics of these datasets, a comparative study of various categorization techniques is performed. In order to evaluate their effectiveness in various diverse circumstances, this study uses models including Logistic Regression, Support Vector Machines (SVM), and Random Forest. A thorough evaluation of predicted accuracy, robustness, and generalization across both datasets is ensured by the use of known measures in the evaluation of these models. In doing so, we hope to learn more about how various models adjust to various data distributions and difficulties while also determining the best categorization strategies for every dataset.

2 Datasets description

2.1 Insurance Dataset (CoIL Challenge 2000)

The first dataset [1] is a small-sized collection of insurance data, focusing on applicants' information. Each record comprises **86 attributes**, categorized as follows:

- **Sociodemographic data (attributes 1–43)**: Derived from zip codes, meaning all individuals within the same zip code share identical sociodemographic characteristics.
- **Product ownership data (attributes 44–86)**: Indicating ownership of various insurance products.

The **target variable** is attribute 86, labeled as *"CARAVAN: Number of mobile home policies"*, representing whether an individual owns mobile home insurance.

2.1.1 Dataset Properties

- **Number of columns:** 86
- **Number of records:** 5822 (train + validation)
- **Particularities:**
  - Data is split into separate files (*train + val, test*).
  - The dataset is labeled

2.2 Credit Card Approval Prediction (Kaggle)

This dataset [2] involves **credit card applicant data** used to predict whether an applicant is classified as a "good" or "bad" client. Traditionally, credit scoring models have been built using **Logistic Regression** for its transparency and binary classification capabilities. However, advancements in **machine learning techniques** such as Boosting, Random Forest, and Support Vector Machines (SVM) have significantly enhanced predictive accuracy. Despite these improvements, complex models often lack interpretability, making them challenging for customer understanding and regulatory compliance.

The task associated with this dataset requires **constructing a machine learning model** to classify applicants while addressing key challenges:

- Undefined "good" or "bad" labels, requiring analytical techniques such as *vintage analysis* for classification.
- Significant class imbalance, necessitating methods like resampling or class weighting to ensure robust predictions.

2.2.1 Dataset Properties

- **Number of columns:** 18 + 3
- **Number of records:** 438,510
- **Particularities:**
  - Dataset is split into two separate `.csv` files.
  - The dataset is **unlabeled**, requiring label assignment methods.

3 Theoretical foundations

3.1 Logistic Regression

Logistic Regression is a statistical and machine learning algorithm used for **binary classification**. It models the probability that a given input belongs to a particular class using the logistic (**sigmoid**) function, which maps any real-valued number to a range between 0 and 1.

3.1.1 Mathematical formulation

Given an input feature vector $X = (x_1, x_2, ..., x_D)$, Logistic Regression models the probability of the positive class ($y = 1$) as:

$$P(y = 1|X) = \sigma(w_0 + w_1x_1 + w_2x_2 + ... + w_Dx_D) \tag{1}$$

where w_0 is the **bias term** and $w_1, w_2, ..., w_D$ are the **feature weights**. The **sigmoid function** $\sigma(z)$ is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

where z is the **linear combination of features and weights**:

$$z = w_0 + \sum_{i=1}^{D} w_i x_i \tag{3}$$

The model predicts class **1** if $P(y = 1|X) > 0.5$, otherwise class **0**.

3.1.2 Loss Function

The **Log-Likelihood function** is used to estimate the weights by maximizing the probability of correct classification:

$$\mathcal{L}(w) = \sum_{i=1}^{N} [y_i \log P(y_i|X_i) + (1 - y_i) \log(1 - P(y_i|X_i))] \tag{4}$$

Since in our implementation we use optimization methods like **Gradient Descent**, we minimize the **Negative Log-Likelihood**, also known as the **Binary Cross-Entropy Loss**:

$$J(w) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log \sigma(z_i) + (1 - y_i) \log(1 - \sigma(z_i))] \tag{5}$$

3.1.3 Gradient Descent Optimization

To find the optimal weights, we compute the gradient of the loss function:

$$\frac{\partial J}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} (\sigma(z_i) - y_i) x_{ij} \tag{6}$$

Weights are updated iteratively using **Gradient Descent**:

$$w_j := w_j - \alpha \frac{\partial J}{\partial w_j} \tag{7}$$

where:

- α is the learning rate,
- $\frac{\partial J}{\partial w_j}$ is the gradient.

3.1.4 Decision Boundary

The decision boundary is defined where:

$$w_0 + w_1 x_1 + ... + w_D x_D = 0 \tag{8}$$

which represents a linear separator in the feature space.

3.2 Soft-Margin Support Vector Machine (SVM)

A **Soft-Margin Support Vector Machine (SVM)** is a supervised machine learning algorithm used for **binary classification**. Unlike Hard-Margin SVM, which requires perfectly linearly separable data, the Soft-Margin SVM introduces a **relaxation** to handle cases where some misclassification is allowed. This is done by introducing **slack variables** ξ_i that allow certain points to be within the margin or misclassified.

The goal remains to find an optimal hyperplane that maximizes the margin while minimizing misclassification errors through a **regularization parameter** C.

3.2.1 Mathematical Formulation

Given a dataset with N samples:

$$(X_i, y_i) \quad \text{where} \quad X_i \in R^D, \quad y_i \in \{-1, 1\}, \quad i = 1, 2, ..., N$$

the **decision boundary** is defined by the hyperplane:

$$w^T X + b = 0 \tag{9}$$

where:

- w is the **weight vector** (normal to the hyperplane),

- b is the **bias term**.

For Soft-Margin SVM, the constraints allow misclassification through slack variables $\xi_i \geq 0$:

$$y_i(w^T X_i + b) \geq 1 - \xi_i, \quad \forall i = 1, 2, ..., N \tag{10}$$

where ξ_i represents the degree of margin violation.

3.2.2 Optimization Problem

The optimization problem for Soft-Margin SVM is:

$$\min_{w,b,\xi} \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} \xi_i \tag{11}$$

subject to:

$$y_i(w^T X_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \tag{12}$$

where:

- C is the **regularization parameter**, controlling the trade-off between maximizing the margin and minimizing misclassification.

3.2.3 Lagrangian Dual Formulation

Using **Lagrange multipliers** $\alpha_i \geq 0$, the problem is reformulated as:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{N} \alpha_i \left[y_i(w^T X_i + b) - 1 + \xi_i\right] \tag{13}$$

Differentiating with respect to w and b:

$$w = \sum_{i=1}^{N} \alpha_i y_i X_i \tag{14}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{15}$$

This leads to the **dual optimization problem**:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(X_i, X_j) \tag{16}$$

subject to:

$$\sum_{i=1}^{N} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad \forall i \tag{17}$$

where $K(X_i, X_j)$ is the **kernel function** (for linear SVM, $K(X_i, X_j) = X_i^T X_j$).

3.2.4 Decision Function

Once the optimization is solved, the final decision function for classification is:

$$f(X) = \sum_{i=1}^{N} \alpha_i y_i K(X_i, X) + b \tag{18}$$

The classification decision is made as:

$$\hat{y} = \text{sign}(f(X)) \tag{19}$$

3.2.5 Kernel Trick

The kernel trick allows SVMs to operate in a higher-dimensional feature space without explicitly computing transformations. Common kernels include:

- **Linear Kernel**: $K(X_i, X_j) = X_i^T X_j$
- **Polynomial Kernel**: $K(X_i, X_j) = (X_i^T X_j + c)^d$
- **Gaussian (RBF) Kernel**: $K(X_i, X_j) = \exp(-\gamma ||X_i - X_j||^2)$

3.3 Random Forest

Random Forest is an **ensemble learning method** used for classification and regression tasks. It builds multiple decision trees and aggregates their predictions to improve accuracy and reduce **overfitting**. Each tree is trained on a random subset of data and features to enhance model robustness.

3.3.1 Mathematical Formulation

A **Random Forest** consists of multiple **Decision Trees**. For a dataset with N training samples:

$$(X_i, y_i), \quad X_i \in R^D, \quad y_i \in \{0, 1\}, \quad i = 1, 2, \ldots, N$$

where X_i represents feature vectors and y_i represents binary class labels.

Step 1: Bagging (Bootstrap Aggregation) To introduce diversity, each tree is trained on a **bootstrap sample**:

- *Random Sampling with Replacement:* Each tree is trained on a subset $X^{(t)}$ of the original data, where some samples appear multiple times and others are left out.

- *Feature Randomness:* For each tree, a random subset of M features (out of D) is selected for splitting at each node.

Step 2: Decision Tree Growth Each tree is constructed using the best splits based on **Gini Impurity** or **Entropy**.

1. Gini Impurity:

$$G = 1 - \sum_{c} p_c^2$$

where p_c is the proportion of class c at a given node.

2. Entropy:

$$H = -\sum_{c} p_c \log_2(p_c)$$

A split is chosen to maximize **Information Gain:**

$$IG = H_{\text{parent}} - \left(\frac{|X_{\text{left}}|}{|X|} H_{\text{left}} + \frac{|X_{\text{right}}|}{|X|} H_{\text{right}} \right)$$

The tree grows until a stopping criterion (max depth, minimum samples per leaf) is met.

Step 3: Aggregating Predictions For classification, each tree votes on the class, and the **majority vote** determines the final prediction:

$$\hat{y} = \{T_1(X), T_2(X), ..., T_B(X)\} \tag{20}$$

For regression, the trees output a numerical value, and the final prediction is the average:

$$\hat{y} = \frac{1}{B} \sum_{t=1}^{B} T_t(X) \tag{21}$$

where B is the number of trees in the forest.

3.3.2 Feature Importance

Random Forest provides **feature importance scores** by measuring how much each feature reduces impurity across all trees:

$$\text{Importance}(f) = \sum_{\text{splits on } f} \frac{\text{Information Gain}}{\text{Total Splits}}$$

4 Design and implementation

The experiments were conducted in **Google Colab**, a cloud-based environment offering accessibility and computational resources. The implementations were adapted from the **ML laboratory** [3], allowing for hands-on development and ease of following structured instructions. Standard Python modules were used to calculate model evaluation and performance indicators, guaranteeing a reliable and repeatable evaluation.

The implementation process was iterative, with multiple trials for each dataset. For the **small dataset**, the first approach involved incorporating categorical features, followed by a second iteration where low-correlation columns were dropped. Finally, a third experiment was performed using the original dataset. Similarly, the **big dataset** underwent structured experimentation, initially inserting categorical columns into the evaluation process. This was followed by combining features based on correlation scores, and the final iteration refined the process by integrating lessons from the small dataset while applying the improved feature engineering.

4.1 Datasets Particularities

The **small dataset** contains only numerical values, but a separate dictionary was provided for categorical column mappings. A notable challenge was the highly imbalanced target variable distribution, which influenced model evaluation. Furthermore, a high number of correlated columns (46 pairs with a correlation grade greater than 0.5) required careful handling during preprocessing to reduce redundancy.

The **big dataset** introduced additional complexities, including missing values in one of its features and structural inconsistencies such as duplicates and mismatched ID counts across the two provided CSV files. Additionally, since this dataset lacked labels, **pseudo-labeling** was implemented to enable model training and evaluation. These factors required meticulous preprocessing and validation to ensure meaningful results.

5 Evaluation

For this project, besides the basic evaluation metrics integrated within the algorithm implementations, such as **accuracy** and **loss**, additional performance metrics were calculated using the **Scikit-learn** library. These metrics provide deeper insight into model effectiveness. The included evaluation metrics are:

- **Precision**: Measures the proportion of correctly predicted positive observations out of all predicted positive observations.

$$Precision = \frac{TP}{TP + FP} \tag{22}$$

  where:

  - TP = True Positives (correctly classified positive cases)
  - FP = False Positives (incorrectly classified positive cases)

- **Recall**: Measures the proportion of correctly predicted positive observations out of all actual positive observations.

$$Recall = \frac{TP}{TP + FN} \tag{23}$$

where:

  - TP = True Positives
  - FN = False Negatives (missed positive cases)

- **F1-Score**: The harmonic mean of Precision and Recall, balancing the trade-off between them.

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{24}$$

- **Confusion Matrix**: A tabular representation of model performance, summarizing predictions against actual values. It consists of:

  - True Positives (TP)
  - False Positives (FP)
  - True Negatives (TN)
  - False Negatives (FN)

The obtained results for both datasets are shown below:

```
<ipython-input-43-5c94959d4550>:10: RuntimeWarning: ove
  out = 1 / (1 + np.exp(-x))
Validation Loss: -0.4789, Validation Accuracy: 0.0000
```

Figure 1: First issue: weird accuracy for small dataset

```
Target distribution in training set: [4373  283]
Target distribution in validation set: [1100   65]
Target distribution in test set: [3762  238]
```

Figure 2: Target distribution - small dataset

```
Validation Loss: 3.7704, Validation Accuracy: 0.8052
(4000, 90)
(4000,)
```

Figure 3: First successful run on Logistic Regression - small dataset

```
<ipython-input-105-ab002da41191>:10: RuntimeWarning: overflow encountered in exp
  out = 1 / (1 + np.exp(-x))
Validation Loss: 3.6342, Validation Accuracy: 0.8223
(4000, 90)
(4000,)
Evaluation Loss: 3.7531, Evaluation Accuracy: 0.7598
```

Figure 4: Second try on Logistic Regression - small dataset

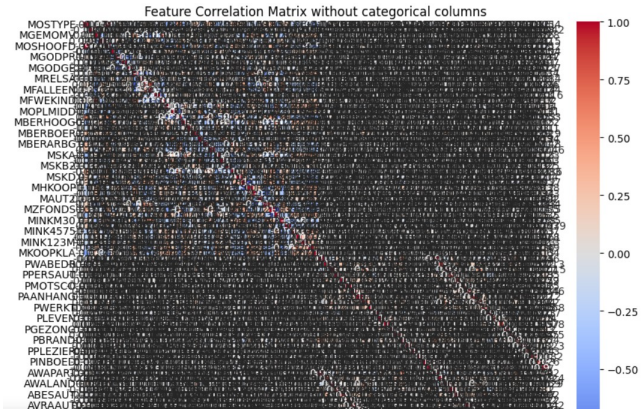

Figure 5: Correlation between columns - small dataset

```
Evaluation Accuracy: 94.05%
Evaluation Set Classification Report:
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      3762
           1       0.00      0.00      0.00       238

    accuracy                           0.94      4000
   macro avg       0.47      0.50      0.48      4000
weighted avg       0.88      0.94      0.91      4000
```

Figure 6: Final run on Soft-Margin SVM - small dataset

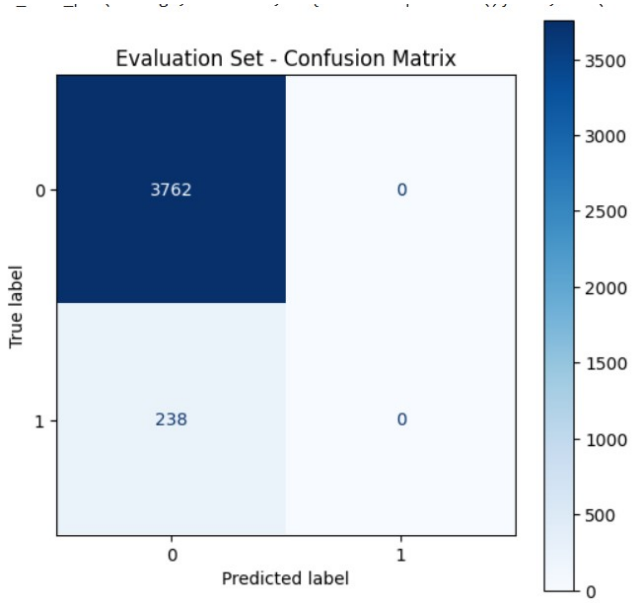

Figure 7: Final run on Soft-Margin SVM - small dataset

```
Evaluation Accuracy: 93.97%
Evaluation Set Classification Report:
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      3762
           1       0.00      0.00      0.00       238

    accuracy                           0.94      4000
   macro avg       0.47      0.50      0.48      4000
weighted avg       0.88      0.94      0.91      4000
```

Figure 8: Final run on Random Forest - small dataset

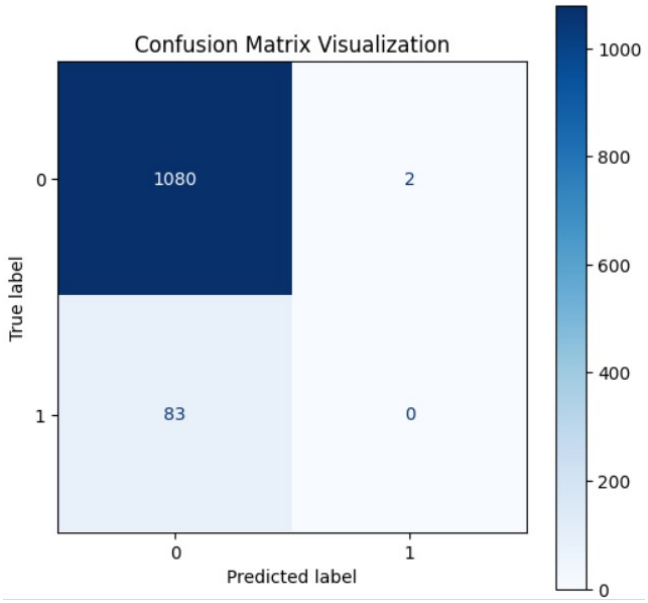

Figure 9: Final run on Random Forest - small dataset

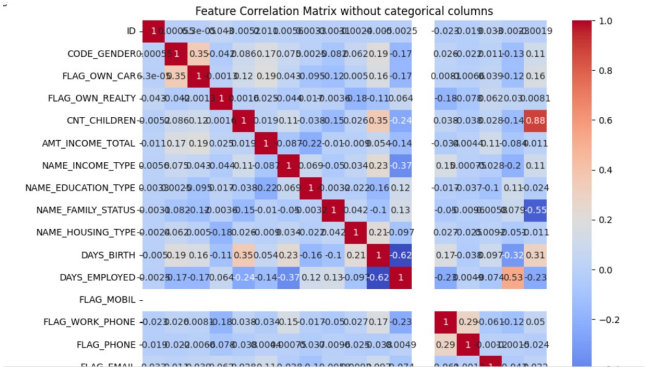

Figure 10: Correlation matrix - big dataset

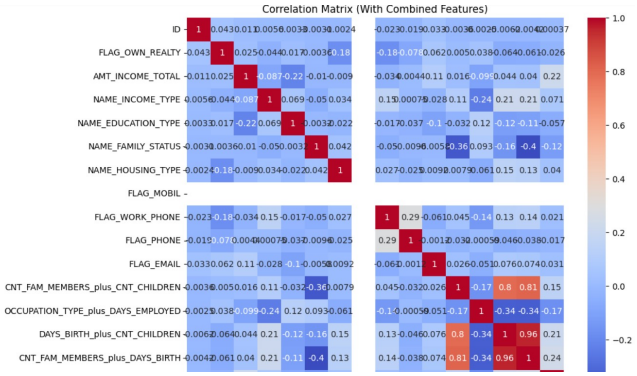

Figure 11: Correlation matrix with combined features - big dataset

```
<ipython-input-55-ab002da41191>:10: RuntimeWarning: overflow encountered in exp
  out = 1 / (1 + np.exp(-x))
Validation Loss: 0.0508, Validation Accuracy: 0.9977
(87712, 17)
(87712,)
```

Figure 12: Final run on Logistic Regression - big dataset

6 Analysis and interpretation

The outcomes for the small dataset demonstrate notable variations across the methods that were tested. Despite having a slow start and a high loss, logistic regression was able to attain good accuracy. However, even after tuning, the loss remained relatively high, and while evaluation accuracy improved, the test dataset revealed a significant increase in loss and a drop in accuracy, indicating potential **overfitting**. Although Soft-Margin SVM had good accuracy, the unbalanced distribution of the target labels caused bias in its predictions, favoring the majority class. Similar problems were found with the Random Forest model, which again had good accuracy but showed bias in favor of the majority class, which limited its capacity for generalization.

For the big dataset, Logistic Regression performed exceptionally well, with very low loss and high accuracy, making it a strong candidate for classification tasks in this context. However, because the kernel's quadratic memory need was more than the computational resources available, Soft-Margin SVM failed to execute correctly, resulting in runtime failure. Despite its theoretical stability, Random Forest had severe computing inefficiency because it ran for more than four hours without yielding any results. These findings imply that, even though logistic regression is a useful and effective option for this dataset, more focused strategies or different SVM implementations would be needed to successfully manage the computational restriction.

7 Conclusions

After evaluating the applied algorithms, we summarize the findings as follows:

For the **small dataset**, Logistic Regression demonstrated good validation performance but significantly deteriorated during evaluation, likely due to overfitting. This observation aligns with the dataset's imbalanced target label distribution. Soft-Margin SVM produced high accuracy on both validation and test sets, but the results appeared biased, as highlighted by the confusion matrix. Random Forest followed a similar pattern, showing high accuracy but a lack of generalization due to class imbalance.

For the **big dataset**, Logistic Regression performed exceptionally well on validation, though evaluation results were not explicitly available. Soft-Margin SVM failed to execute due to its kernel's quadratic memory requirement, making it unsuitable for large datasets. Random Forest remained inconclusive as it continued running beyond feasible time limits. A key consideration for this dataset is that all testing was conducted using pseudo-labeling due to the absence of predefined labels.

These conclusions highlight the strengths and limitations of each algorithm, providing insights into their applicability for different dataset characteristics.

8 Future work

There are multiple opportunities for further improvement of this study:

- **Fine-tuning parameters:** The presented results were obtained using a predefined, rigid setup, primarily based on laboratory exercises. Each algorithm's performance can be enhanced by optimizing its hyperparameters through techniques such as grid search or Bayesian optimization.
- **Data quality and challenges:** As discussed earlier, dataset-specific characteristics significantly impact the performance of the models. For instance, in the first dataset, class imbalance likely contributed to the observed overfitting in Logistic Regression. In the second dataset, all predictions relied on pseudo-labeling, which might introduce biases.
- **Handling scalability:** The Soft-Margin SVM failed to execute on the larger dataset due to its quadratic memory complexity. One possible solution is dimensionality reduction through techniques like Principal Component Analysis (PCA), which can help manage computational costs.
- **Exploring advanced algorithms:** For larger datasets, traditional models may be insufficient due to computational constraints. Implementing deep learning techniques, ensemble models, or tree-based boosting methods (such as XGBoost) could improve efficiency and predictive performance.

By addressing these aspects, we can further refine the performance, scalability, and robustness of the classification models for diverse datasets.

References

[1] First dataset: *Insurance Company Benchmark (CoIL 2000)*, https://archive.ics.uci.edu/dataset/125/insurance+company+benchmark+coil+2000.

[2] Second dataset: *Credit Card Approval Prediction (Kaggle)*, https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction.

[3] ML laboratory : https://moodle.cs.utcluj.ro/course/view.php?id=697.