Computational Intelligence II - Project 2

Pedro Henrique Pamplona Savarese - phpsavarese@gmail.com

Code: https://github.com/lolemacs/heart-attack-prediction

Introduction

The goal of this project is to construct and compare Machine Learning models on a toy example. For that, we will cover several aspects of a general Machine Learning pipeline, including feature analysis, model construction, hyperparameter optimization and performance measurement.

Our problem is defined as a binary classification task with only 3 features: total cholesterol, age and glycemia level. Each data point corresponds to a past patient, and the label denotes if the given patient had cardiovascular complications.

The training set is composed of only 874 patients and is considerably unbalanced. The features' statistics are:

	Mean	Std
Total Cholesterol	210.2	31.5
Age	45.1	18.6
Glycemia Level	114.6	27.1

Out of the 874 patients, a total of 291 encountered cardiovascular complications (are labeled as 1) and 583 haven't. Therefore the class proportions are 0.33 and 0.67, respectively. This will play an important role in the project since training a model to maximize its accuracy would most certainly result in a bias towards the 0 class - meaning that patients which are likely to have cardiovascular issues in the future would be given less importance.

As an example to show how unbalanced classes are a key aspect when training models, suppose we have a set of patients and we want to train a model to diagnose a rare disease - so rare that its occurrence is 1 out of 1.000.000 patients. A model that ignores all the patient's features and automatically classifies it as healthy would have an expected accuracy of $\frac{999.999}{1.000.000} = 0.999999$. One could say that this is a good model due to its extremely high accuracy, however this model would also misdiagnose all sick patients, therefore possibly killing all of them (in case the disease in question is lethal).

To approach this problem, we will compare several famous models in the Machine Learning field.

Methodology

For each model, we will split the dataset in two parts: a training set and a validation set. When cross-validation isn't used, the split will be performed as follows: first the dataset is shuffled, then 80% of the data points are used as training set and the remaining 20% as test set. For the models trained with cross-validation, 5 folds will be used.

The metric chosen to measure each model's performance is the F1-Score. It is defined as the harmonic mean of Precision and Recall. To further understand each of these, we present the confusion matrix:

	Prediction = 1	Prediction = 0
Label = 1	True Positive	False Negative
Label = 0	False Positive	True Negative

Precision and Recall are defined as follows:

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

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

And finally:

$$F1 = 2\frac{Precision.Recall}{Precision + Recall}$$

Using the rare disease example once again, a model that classifies all patients as healthy ($Prediction = 0$) would have 0 True Positives and False Positives. Therefore its $F1$ score would be 0, showing how bad such a model would really be.

A good way to take the $F1$ score in consideration is not only to use it as a comparison measure, but also to force the model to take into account how the classes are unbalanced. For that, setting up different class weights whenever possible is an usual practice in Machine Learning.

For the models that have the ability to incorporate class weights during their training, we will use weights inversely proportional to their frequencies:

$$weight(y_k) \propto \frac{N}{\sum_{i=1}^{N} \mathbb{1}\{y_i = y_k\}}$$

For example, if a dataset has a total of 100 data points with label 0 and 900 with label 1, we could set $w_0 = 10$ and $w_1 = 1.1$. This would make the model care roughly 10 times more about data points with label 0 than with label 1.

To optimize hyperparameters, we will use the validation set (or cross-validation, whenever applicable) as a surrogate for the test set performance. That is, for each set of hyperpameters $S_i \in \{S_1, S_2, \ldots, S_m\}$, a correspondent performance on the validation set can be directly calculated: $P_{val}(S_i)$. Choosing a set of hyperparameters is then as easy as:

$$S_{best} = \underset{S_i}{\operatorname{argmax}}(P_{val}(S_i))$$

Results

In this section, we will present the results on all tested models.

Bayesian Ridge Regression

Grid search was used to optimize $\alpha_1, \alpha_2, \lambda_1$ and λ_2 and the features were expanded up to 3-rd degree powers. The search space for each parameter was first set to $\{1^{-8}, 1^{-7}, 1^{-6}, 1^{-5}, 1^{-4}\}$ and no class weights were used. For each grid search result, the search space would be expanded with $1^{i-2}, 1^{i-1}$ if the best value was the lowest on the grid, and with $1^{i+1}, 1^{i+2}$ if it was the highest.

The best set of parameters found using cross-validation was $\alpha_1 = 1^2, \alpha_2 = 1^{-12}, \lambda_1 = 1^{-13}$ and $\lambda_2 = 1^{-1}$. The resultant $F1$ score was 0.63.

Random Forests

Grid search was used to optimize the maximum number of features per decision tree $maxFeat$, the maximum depth of each tree $maxDepth$ and the number of trees in the ensemble $nTrees$. The features were expanded up to 10-th degree powers. The search space for each parameter was first set to $\{1, 2, 3, 4, 5, 6\}$, except for $nTrees$ which was $\{100, 200, 300, 400, 500, 600\}$.

Class weights were used. For each grid search result, the search space would be expanded with $1^{i-2}, 1^{i-1}$ if the best value was the lowest on the grid, and with $1^{i+1}, 1^{i+2}$ if it was the highest.

The best set of parameters found using cross-validation was $maxFeat = 5, maxDepth = 4$ and $nTrees = 500$. The resultant $F1$ score was 0.67.

Logistic Regression

Grid search was used to optimize the regularization strength C. The features were expanded up to 3-rd degree powers. The search space for C was $\{1^{-2}, 1^{-1}, 1^0, 1^1, 1^2, , 1^3\}$.

Class weights were used. For each grid search result, the search space would be expanded with $1^{i-2}, 1^{i-1}$ if the best value was the lowest on the grid, and with $1^{i+1}, 1^{i+2}$ if it was the highest.

The best set of parameters found using cross-validation was $C = 1.0$. The resultant $F1$ score was 0.65.

Gaussian Naive Bayes

No grid search was used since this is a parameter-free model. The validation set was used only to find up to which power the features were expanded to. The value found was 3.

The resultant $F1$ score was 0.61.

Nearest Neighbors

Grid search was used to optimize the number of neighbors k, how neighbors are weighted according to their distances and the power for the Minkowski distance p. The search space for k was $\{1, 2, 3, 4, 5, 6, 7, 8\}$, for p $\{1, 2\}$ and the weighting could be either uniform or proportional to the neighbor's distance.

For each grid search result, the search space would be expanded with $1^{i-2}, 1^{i-1}$ if the best value was the lowest on the grid, and with $1^{i+1}, 1^{i+2}$ if it was the highest.

The best set of parameters found using cross-validation was $k = 8, p = 2$ and weights proportional to distances. The resultant $F1$ score was 0.58.

Support Vector Machines

Grid search was used to optimize the kernel type, the penalty parameter C and the polynomial degree d for the polynomial kernel. The search space for C was $\{0.001, 0.1, 1.0, 10.0, 100.0\}$, for d $\{1, 2, 3, 4, 5\}$ and possible kernels were linear, RBF and polynomial.

The best set of parameters found using cross-validation was $C = 0.1, d = 1$ with the polynomial kernel. The resultant $F1$ score was 0.65.

Residual Networks

For this model grid search wasn't used due to the enormous space and number of hyperparameters. Instead, strong regularization techniques were used coupled with early stopping.

The final architecture is composed of 7 layers: first a linear layer with 500 hidden neurons, followed by 5 hidden residual ReLU layers, all with 500 neurons, 0.2 dropout and batch normalization. Lastly, a batch-normalized sigmoid layer outputs the predicted labels.

L2 penalization was used on the γ parameters of the batch normalization layers in the main residual block only. No other parameters suffered penalization.

The features were expanded up to their 10th powers, resulting in 286 input features. All weights were initialized with He initialization. Class weights were used and the optimization technique was Adam with nesterov momentum.

After 19 epochs, the model achieved its best $F1$ score on the validation set: 0.69.

Results

	F1
Bayesian Ridge Regression	0.63
Random Forests	0.67
Logistic Regression	0.65
Gaussian Naive Bayes	0.61
Nearest Neighbors	0.58
SVM	0.65
Residual Network	0.69

Discussion

As we've seen, the best models were Residual Networks and Random Forests, showing how boosting techniques can be important when tackling classification problems in Machine Learning. We've also seen principles on how to use cross-validation, hyper-parameter optimization via grid search and a bit on choosing the right metric for the problem.

By seeing the results from SVMs and Logistic Regression we also see that the problem isn't defined by a difficult separation surface: for SVMs, a polynomial kernel without degree expansion was the one that performed best on CV, while Logistic Regression is good for linearly separable classes. These two models had an $F1$ score not that below the best one: 0.65 versus 0.69.