Report: Kaggle challenge. Classification of extreme weather events

Frida-Cecilia Acosta-Parenteau
Université de Montréal
Montréal, QC, H2P 1K4
frida-cecilia.acosta-parenteau@mail.mcgill.ca

1 Introduction

In this Kaggle challenge, we had the task to classify extreme weather events using a multi-class logistic regression, implemented using numpy and pandas. As well as any other model taken from sci-kit learn to compare the performance of predictions. I chose to use the XGBoost classifier from scikit-learn library.

The dataset consists of 19 features. 16 being meteorological data and three others being the time, the longitude and the latitude. The dataset for training comprises 44,760 observations recorded between 1996 and 2009, while the dataset reserved for testing encompasses 10,320 observations spanning from 2010 to 2013.

The main challenge of this Kaggle challenge is that the extreme weather events, which tropical cyclone (label: 1), or atmospheric river (label: 2) have a much smaller sample size compared to the background weather (label: 0).

2 Feature Engineering

I did some data transformations to be able to have better performance on the model. First I normalized the features using the min-max normalization. I also changed the 'date' feature make it cyclical (using sin and cos) in terms of days of the year, since we want to use it to extract the information about the time of the year because the climate change as a function of seasons. Finally we removed the duplicated rows, since they pollute the data.

Furthermore, we have a class imbalance in the labels of the three classes. Indeed, the counts for each class are the following: $\{0 : 17099, 2 : 4250, 1 : 1033\}$.

3 Algorithms

3.1 Logistic Regression

Logistic regression is one of the most used model for classification as it is a simple model that can be useful on datasets with a lot of features (1). I implemented the logistic regression classifier, in one-vs-all. Therefore for each class, a separate logistic regression classifier is trained to distinguish that class against all other classes. For the model that is to distinguish class 1 from the rest, the target variable in the training data is modified so that it's 1 if the original class was 1 and 0 otherwise with the help of one-hot encoding. It uses Gradient Descent to iteratively adjust the parameters (weights and bias) to minimize the cross the cross-entropy loss across all observations in the training set. This process is repeated for all classes. For prediction, for each new input instance, each classifier (for label 0, 1, 2) predicts a probability, which is calculated using the softmax, that the instance belongs to its class. The class corresponding to the classifier with the highest probability function is taken as the prediction.

3.2 XGBoost

XGBoost is based on the gradient boosting framework. Gradient boosting is an ensemble technique that builds a model in a stage-wise fashion. More specifically it consists of a set of decision classification and regression trees (CART). A CART is a bit different from decision trees, in which the leaf only contains decision values. In CART, a real score is associated with each of the leaves, which gives us richer interpretations that go beyond classification. It constructs a large number of trees in a sequential manner, where each tree tries to correct the mistakes of its predecessor. It constructs new models that predict the residuals or errors of prior models and then combines them to make the final prediction.

The individual models in XGBoost are decision trees. A key feature of XGBoost is the inclusion of regularization terms in the loss function, which helps prevent overfitting. It has a number of parameters that can be fine-tuned to optimize performance, including the depth of the trees, learning rate, and the number of trees to construct. (2)

The objective function of the XGBoost algorithm can be expressed as follows:

$$\text{Obj}(\Theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{1}$$

where:

- $l(y_i, \hat{y}_i)$ is a differentiable convex loss function that measures the difference between the predicted value $\hat{y}_i$ and the actual value y_i.
- $\Omega(f_k)$ is the regularization term which penalizes the complexity of the model. It is defined as:

  $$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|\omega\|^2 \tag{2}$$

  where T is the number of leaves in the tree, ω is the vector of scores on the leaves, γ is the complexity control on the number of leaves, and λ is the L2 regularization term on the weights.
- $\hat{y}_i^{(t)}$ is the prediction for the i-th instance at the t-th iteration, which is updated as follows:

  $$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i) \tag{3}$$

  where $f_t(x_i)$ represents the prediction of the t-th tree.
- The function $q(x)$ maps features of instances to leaves in the tree:

  $$f_t(x) = w_{q(x)}, \quad q : \mathcal{X} \to \mathcal{T}, \quad w \in \mathbb{R}^T \tag{4}$$

4 Methodology

4.1 Logistic regression mini batch

First I implemented a logistic regression that performed gradient descent using the full batch (full dataset). After testing full-batch and realizing that the model was converging too slowly as seen in the following figures 1a, 1b for 1000 iterations and 2000 iterations while training on the full training set. We can clearly see on this figure that the gradient descent is very unstable and that it is far from having converged after 1000 iteration steps. Thus, I did a mini-batch gradient descent instead. Each mini batch was sampled with replacement and use stratified sampling to be able to train on more underrepresented data. For each class label, the stratified sampling proportions were changed, as seen on 1.

Changing the sampling proportions was part of hyper-parameter tuning. Unfortunately I could not implement a full grid-search that could combine multiple learning rate, stopping criterion, regularization techniques and values an finally batch size, due to computing and time constraints on my personal computer. Therefore, I had to test learning rates by themselves. I chose to use the time-decaying learning rate which was $lr = 1/1 + k$ were k was the iteration steps of the gradient descent (3). I had tested other learning rates that were not decaying but they did not converge well and added a lot of noise and stochastic behavior. I decided to test multiple regularization combinations. Using L1, L2 and elastic-net. I fine-tuned parameters through multiple experimentation measuring f1-score and accuracy and chose these final hyper-parameters to test on the validation set with elastics net: $\lambda = [0.01, 0.1, 1], \alpha = [0.5, 0.75, 0.9, 1]$. With the formula Elastic net $= \alpha\ \lambda\ * L1 + (1 - \alpha\ )\lambda\ * L2$. The early stopping criterion chosen was:

$$|\text{prev\_loss} - \text{loss}| < \epsilon \quad \text{and} \quad k > 100$$

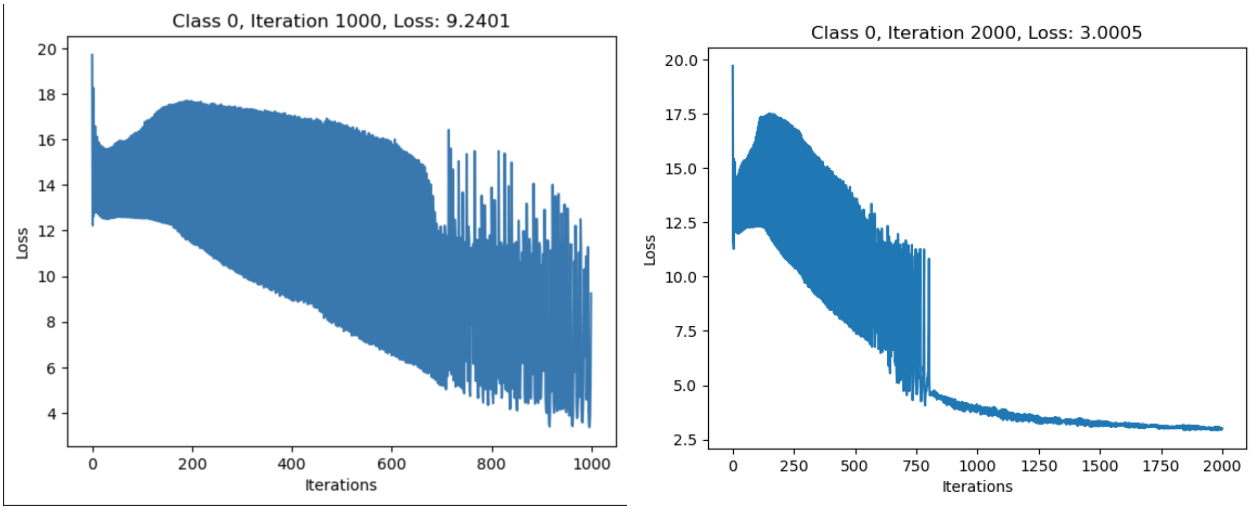
I also did cross-validation but only with two splits due to time and power constraints with my computer.

4.2 XGBoost classifier from Scikit Learn

I used the XBoost Classifier from Scikit Learn. I used the SMOTETomek resampling technique also from the same library, which is doing SMOTE and Tomeklinks resampling. The training data is resampled to adjust the class distribution. The sampling_strategy parameter is set to oversample the minority class to 2000 instances for class 1 and 5000 instances for class 2.

	class 0	class 1	class 2
class 0	0.7	0.10	0.20
class 1	0.6	0.2	0.2
class 2	0.6	0.15	0.15

Table 1: Different distributions of the classes to create the mini batches during the training of each class. Each row represent a distribution



(a) 1000 iterations

(b) 2000 iterations

Figure 1: Cross entropy loss values during the training of the Background class

For the XGBoost Classifier, the eval_metric is set to "mlogloss" to evaluate multiclass log loss during training. A dictionary of hyperparameters and their possible values is defined for the XGBoost model, including learning_rate, n_estimators, max_depth, subsample, colsample_bytree, and gamma. For hyper-parameter tuning, I used the RandomizedSearchCV object. It randomly selects 50 combinations of parameters to try, using 3-fold cross-validation and accuracy as the scoring metric. Then I used the result from the best model, that was chosen according to accuracy.

5 Results

5.1 Full batch logistic regression

We can see the results for the full batch logistic regression in Figure 2 obtained from the cross-validation for different values of λ. However, the best accuracy obtained on the test was 71% with the full-batch.

5.2 Mini-batch logistic regression

We can see the results of an experiment where I used Logistic regression with mini-batch, with no early stopping and sample_per_batch=35. L2 regularization : 0.0001. Learning rate at 1/(1+steps) in the Figure 3b. We can see that the weighted F1 score is a lot better than the macro F1-score, because it is more influenced by class 0, since it has a bigger weight and class 0 has a good precision. We can see that the lambda values between 10^{-5} and 10^{-1} had the best results in F1-score. The best accuracy I got on the test set was 72.8%.

Then, I used the best results from some experiments to create a final one to choose the hyper-parameters to use in the test set. I combined different coefficients that change the weight of the regularization strengths for L1 and L2. We can see on Figure 4a and 4b that the best accuracy and mean macro F1-Score was the with the $\alpha = 0.5$ which means L1 and L2 regularization are added with equal proportions and with each $\lambda = 0.1$. It resulted in an accuracy score of 77% from the test.csv dataset, which is a big jump from the previous experiment above.

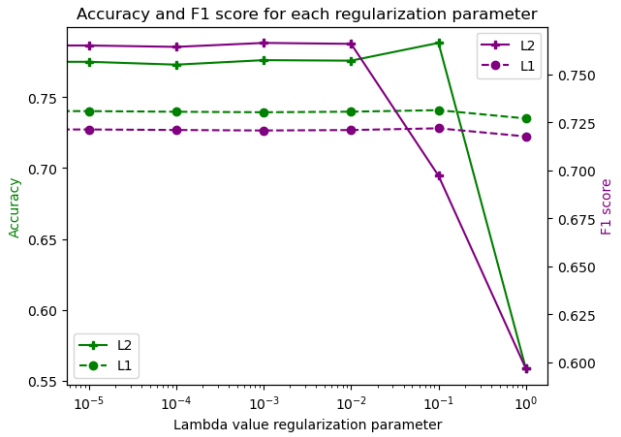
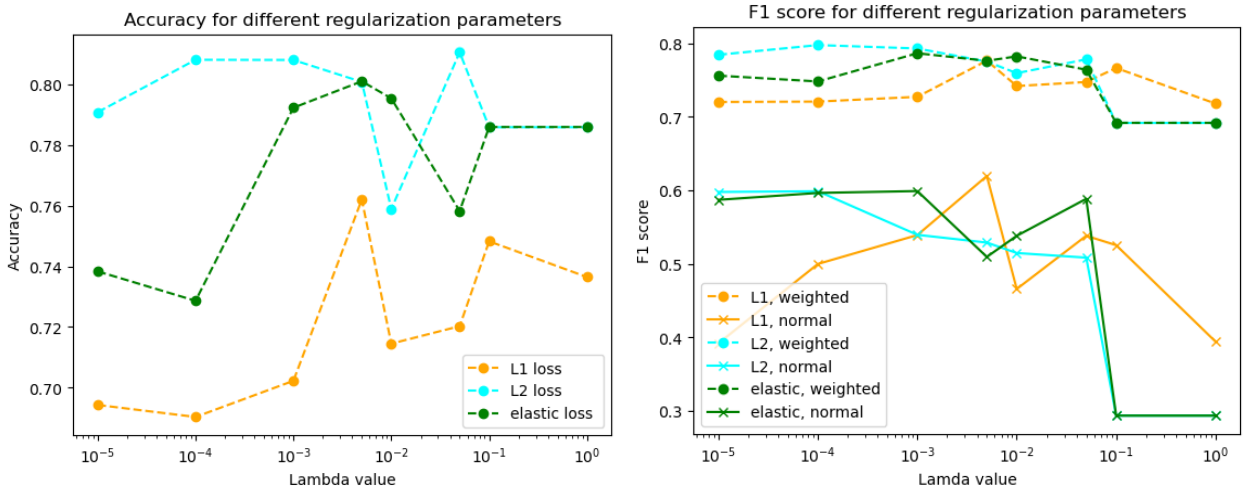


Figure 2: Hyperparameter tuning with accuracy and f1 score metrics for full-batch 1000 iterations



(a) Accuracy

(b) F1 score

Figure 3: Evaluation metrics to evaluate performance for different hyper-parameters in the 4th experiment, where L1, L2 and elastic-net were separated

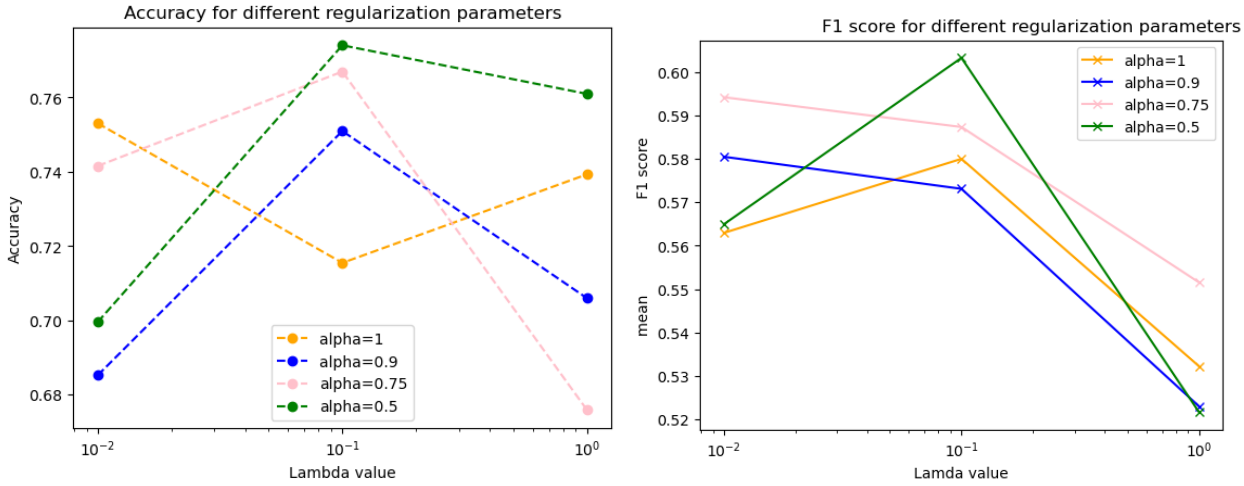
5.3 XGBoost SMOTETomeks resampling

Finally, we have the results for the other model we tested. We got better accuracy results with this model, getting 78.3% on the test dataset, compared to 77% using logistic regression. Most importantly, I had way better results in the validation set on the macro F1-score which is important to identify the performance of the model on the underrepresented classes. The performance on the class 1 and 2, is more important because they are the extreme weather events classes, whereas class 0 is the background class.

Class	Precision	Recall	F1-score	Support
0	0.92	0.88	0.90	3393
1	0.62	0.68	0.65	211
2	0.68	0.76	0.72	873

Table 2: Classification report from the Scikit learn library for the XGBoost classification

We can see from 2, that the class 1 and 2 have a very low precision: 62% and 68% respectively and low recall too. However, when we take the mean, we get 0.76. This is better than the F1-score average for the best hyper-parameter combination model of logistic regression as seen in 4b.



(a) Accuracy (b) F1 score

Figure 4: Evaluation metrics to evaluate performance for different elastic-net hyper-parameters

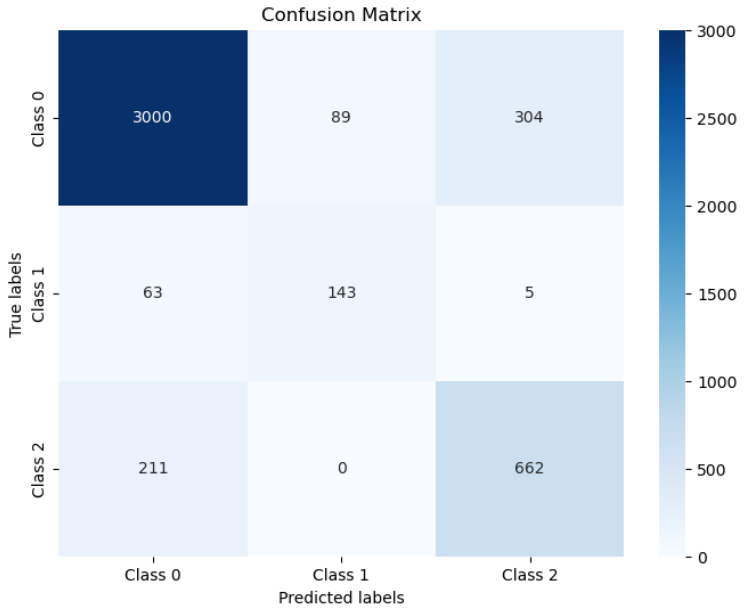


Figure 5: Confusion matrix from the prediction given by the XGBoost classifier

6 Discussion

6.1 Mini-batch logistic regression:

Performance and Generalization full-batch vs. mini-batch.:

In full-batch logistic regression the gradient calculated is precise for the given dataset, but this can lead to overfitting as the model is optimized for the entire dataset at once. For mini-batch, it introduces noise training process since the gradient is computed on a subset of the data. This noise can help the model generalize better by preventing it from fitting too closely to the training data (a form of regularization).

6.2 Logistic Regression vs. XGBoost

XGBoost

XGBoost can capture complex nonlinear relationships between features and the target variable by constructing a series of decision trees, where each tree corrects the errors of the previous one. This is inherently more flexible than the linear decision boundary of logistic regression. It also naturally models feature interactions since the trees split on different features, which is something logistic regression cannot do without explicitly creating interaction terms. In retrospect, I did some mistakes such as using the accuracy as the scoring metric to choose the best hyper-parameter, while I should've chosen F1-score, since we cared more about the performance on the minority classes. (2)

Logistic Regression

To have better predictions I could've done more feature engineering because this model is sensitive to good features and gets better when it is. Also I did not follow a methodical way such as Random grid search to find all the combinations of hyper-parameter, which could've made me miss an optimal set. Since the main challenge of this project was to understand how to have good predictions on the minority classes (1 and 2), one good method I could've used is to use to accommodate the uneven data distribution is applying a specific class weighting strategy during the training phase to alter the degree to which the coefficients of the logistic regression are modified. By implementing this strategy, the algorithm can be fine-tuned to decrease the penalty for misclassifications of the more common class and increase the penalty for the rarer class. Such an approach yields an improved logistic regression model that is better suited for imbalanced classification challenges, often described as weighted or cost-sensitive logistic regression. (4)

7 Statement of Contribution

I hereby state that all the work presented in this report is that of the author.

References

[1] Ashwin Raj, "Title of the medium article," Medium, Month of Publication 2023, Available: `https://thisisashwinraj.medium.com/`.

[2] "Xgboost documentation," Read the Docs, 2023, Accessed: 2023-11-07.

[3] "Learning rate schedules and adaptive learning rate methods for deep learning," Medium, 2023, Accessed: 2023-11-07.

[4] Jason Brownlee, "Cost-sensitive logistic regression for imbalanced classification," Machine Learning Mastery, 2020, Accessed: [insert date here].