Classification of Extreme Weather Events

by Thierry Jean (kaggle: TJeanz, student id: 20127778)

Introduction

The weather is a chaotic system. Small variations in wind directions, humidity, etc. can have a tremendous impact on larger meteorological events. Given the complexity of such system, machine learning proves to be a useful tool for modelling the weather and better understanding it.

Using a subset of the ClimateNet dataset containing 47,760 labeled weather events with 18 variables, classification models were trained to label meteorological events as either: *standard conditions* (class 0), *tropical cyclone* (class 1), or *atmospheric river* (class 2). The performance of the models was assessed using the error rate on a test set containing 7,320 unlabeled events. The original data was preprocessed and used to train various machine learning algorithms including: *Softmax multiclass logistic regression*, *One-vs-one binary logistic regression*, and *XGBoost*. The best performance was achieved by both a Softmax model and an XGBoost model with a test accuracy of 0.76803.

Feature Design

The original dataset consists of a table with one row per event, and one column per feature. The 18 features are of varied nature, and use distinct units. This includes longitude-latitude coordinates, precipitations, wind, humidity, sea level, pressure, and dates. About half of the training set appeared to be duplicated entries (25,763 non-redundant). The duplicates were immediately dropped to avoid them biasing data exploration and feature distributions.

Data exploration

First, a widget allowing to visualize 1D histograms for the distribution of features, and 2D scatter plots for pairwise relationships was used to explore the data. Several insights gained directed the feature preprocessing:

1. Several features appear normally distributed and leptokurtic (U850, V850, UBOT, VBOT, PS, PSL, T200);
2. A few features appear unimodal but skewed (T500, TREFHT, Z100, Z200, ZBOT);
3. TS appears not unimodal;
4. PRECT appears to follow a negative logarithmic distribution, and/or has significant outliers;
5. Longitude and latitude appear as 7 main points on the scatter plot;
6. Events are not uniformally distributed over time.

Feature engineering was done based on the training set only to avoid leakage and keep the test set truly independent. Otherwise, the empirical risk estimate would be biased.

Feature Extraction

[1, 2, 3] For the normally distributed features a *Z-score* standardization was applied. For the skewed data, the square root or the power of 2 standardization was used depending on the side of the skew. TS was also transformed using the square root standardization since it appears highly correlated with TREFHT (another skewed unimodal feature).

[4] For PRECT, the values were scaled up to improve numerical stability. An histogram was build from the log transform of the values. The new feature used the index of the histogram as value, and negative values (n=606) were attributed bin 0.

[5] Based on the visual inspection of the histograms, the 7 locations could be divided into into 2 latitude bins (above and below 30), and 3 longitutde bins (below 260, between 260 and 300, and above 300). The indices of the respective bins were returned as feature value. Since these slices describe seemingly discrete areas, the two columns were one-hot encoded resulting in 5 new columns.

[6] Based on the assumption that the rate of meteorological events varies at various time scales, the year and the month of the dates were encoded as two separate ordinal features.

Algorithms

Three main approaches were tested on this multiclass classification problem.

First, a softmax logistic regression was implemented from scratch. Its weights and biases are optimized through gradient descent using the cross-entropy loss. The model outputs a prediction probability for each class (summing to 1), and the maximum probability class is attributed.

Second, a one-vs-one logistic regression scheme was implemented from scratch. The created model can handle an arbitrary number of label classes. For each pair of classes, it trains a logistic regression (binary classifier) using gradient descent with the binary entropy loss. To generate a prediction, each model outputs a probability for both classes (summing to 1), then the class with the maximum probability is returned.

Third, an extreme gradient boosting (XGBoost) classifier algorithm from the xgboost Python package was used. The objective used multiclass softmax, and the evaluation metric was cross-entropy loss.

Methodology

Along the feature extraction methods mentioned, functions to realize balanced undersampling, to select a subset of features, and to use cross-validation were implemented.

The balanced undersampling process consisted of identifying the class with the least number of examples, and randomly selecting an equal number of examples for each classes. Balanced sampling can help models learn the distinctive properties of each class, and reduce biases towards the majority class. Models were trained with and without balanced undersampling.

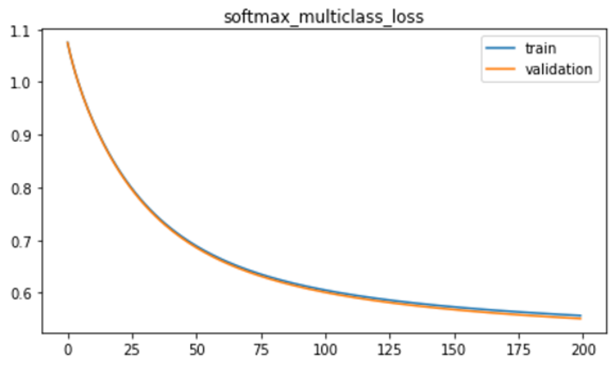
Regarding feature selection, different subsets of features were manually identified based on their covariance and distributions (data exploration). The feature set including only original features normally distributed did not provide good performance. Then, the set including all Z-score standardized features lead to decent performance, and served as a baseline. Adding all hand-crafted features did not appear to significantly improve the models' performance.

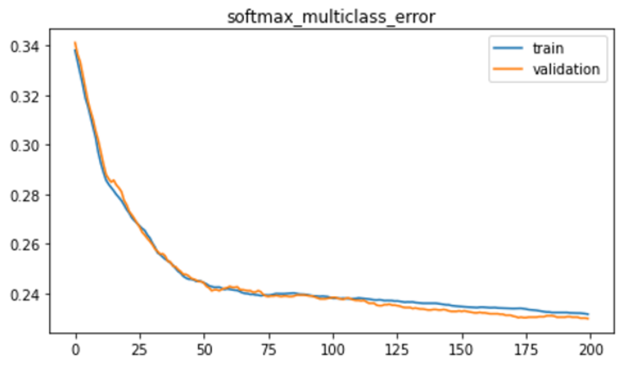
For cross-validation, the method implemented separates the training set according to class labels, then keeps 70% of each class for training, and sets aside the remaining 30% of each class for validation (about the size of the test set). The loss and the error rate were computed on both the training and the validation set. Observing a plateau or an uptrend in any of the two metrics for the validation set guided the hyperparameter selection. Once the hyperparameters were decided, a new model was trained using the full training set to make the final test predictions.

Results

The three types of model share the hyperparameters *number of steps*, which dictates the number of gradient descent iterations, and the *learning rate*, which defines the scale of weight changes at each step. The number of steps should be set to the number of iterations when evaluation metrics start to plateau, or the metrics on the validation set diverges. Such trends in a plot would suggest the model is overfitting. Using lower learning rate can smooth the learning curve, but will require an increased number of steps to converge.

Models were evaluated using their respective loss, and the error rate (i.e., *n correct / n total*, or *1 - accuracy*). Due to the limited number of submissions possible on Kaggle, test performance remains unknown for many preprocessing, features, and model configurations. For each type of model, a table containing statistics for models trained on the same preprocessed dataset is available. The *learning rate* was set at 0.05, and *n_steps* of 30, 70, 100 were tested. The graphs display evaluation metrics up to 200 steps.

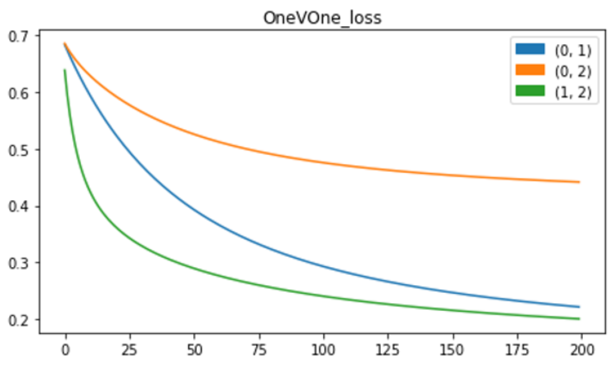


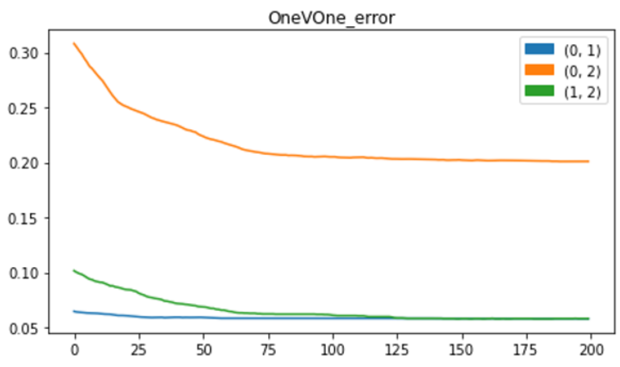


Softmax Multiclass Logistic Regression stats

run	eta	n_steps	class0	class1	class2	train_loss	train_error	val_loss	val_error
0	0.05	30	5992	14	1314	0.7741	0.2623	0.7717	0.261
1	0.05	75	6737	0	583	0.6367	0.2393	0.6331	0.2389
2	0.05	100	6828	0	492	0.6059	0.2383	0.6018	0.2381

Softmax comments: As *n_steps* increases, votes for class 1 disappear, and class 2 diminish. It suggests that model is biased towards the majority class, and has more difficulty distinguishing class 0 and 1 than 0 and 2. The decrease of error rates on the training and validation sets become marginal after 75 steps.

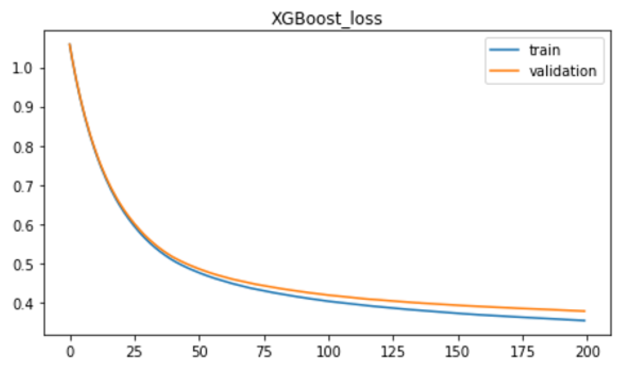


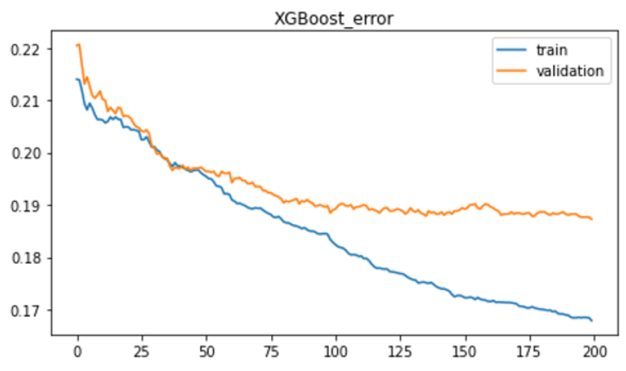


One vs One Logistic Regression stats

run	eta	n_steps	class0	class1	class2	(0, 1) loss	(0, 1) error	(0, 2) loss	(0, 2) error	(1, 2) loss	(1, 2) error
0	0.05	30	4182	746	2392	0.474	0.0589	0.5663	0.2417	0.3311	0.0775
1	0.05	75	5612	70	1638	0.3341	0.0583	0.4957	0.208	0.2612	0.0622
2	0.05	100	5983	34	1303	0.2944	0.0583	0.4759	0.205	0.2414	0.0615

OneVsOne comments: As *n_steps* increases, votes for class 1 and class 2 diminish. Since this ensemble relies on binary classifiers, it might be initially more resilient to class imbalance. Since each binary classifier outputs a probabilistic prediction, an overconfident model will bias the predictions. It is a challenge to train this ensemble without overfitting individual classifiers, since each require a different number of steps to fit the data. For model (0, 1), the error rate plateaus already at 30 steps, and after 75 for models (0, 2) and (1, 2). Considering these metrics are on the training set (no validation set for this model), it is likely to be overfitting.





XGBoost stats

run	eta	n_steps	class0	class1	class2	train_loss	train_error	val_loss	val_error
0	0.05	30	6370	148	802	0.5647	0.201	0.572	0.2012
1	0.05	75	6301	134	885	0.431	0.1884	0.4438	0.1923
2	0.05	100	6281	139	900	0.4042	0.1828	0.4196	0.189

XGBoost comments: As *n_steps* increases, votes for class 0 diminish, class 1 are consistent, and class 2 increase. This distribution of votes more closely ressemble the training set distribution. Since XGBoost relies on building decision trees on the most relevant features, the models appear to better preserve the distinctions between class 0 and class 1. The rate of decrease in evaluation metrics suggests that it could be valuable to train more than 100 steps.

Comparison

The XGBoost seem to achieve the best performance because it better preserves class separation. Indeed, XGBoost models achieve lower loss and error rates than Softmax models. This is likely due to its ability to exploit the most informative features. In contrast, the two other approaches did not include regularization terms, and could have been hurt by redundant or irrelevant features.

Test performance

Because of less than optimal experiment tracking, the full details of the configurations leading to each results is unavailable. The best test accuracy achieved by a Softmax Multiclass Logistic Regression was: 0.76803, for OneVsOne ensemble: 0.74098, and for XGBoost: 0.76803. Even though the type of errors is unknown, it should be noted that Softmax probably achieved this accuracy by never predicting class 1, and being "blind" to it.

Discussion

Knowing the random baseline has an accuracy of 0.703055, and the first place on the leaderboard achieved 0.79836, the best test accuracy from this work (0.76803) is satisfying. While it beats all baselines, it also highlights that further improvements are possible.

Even though some feature distributions were multimodal, skewed, or discontinuous, hand-crafted features did not significantly improve performance. An hypothesis for these results would be that the resulting representations (using bins or ordinal values) were more difficult for gradient descent, and had low correlations to the label. For example, the binned longitude and latitude could have been used as ordinal features instead of being one-hot encoded. More systematic work to remove redundant and irrelevant features would be beneficial. Implementing L1 and L2 penalties could lead to a better optimization by inducing sparsity in features, or penalizing weights having too much pull on predictions.

Finally, since the Kaggle competition format limits the number of performance assessment on the test set, and it is partially hidden, more systematic feature selection and model evaluation would have been beneficial. Adding an hyperparameter grid-search could would help find ideal hyperparameter combinations, especially for models with a larger number of parameters (i.e., XGBoost). In addition, k-fold cross-validation could improve confidence in models by limiting the bias induced by the random train-validation split, and avoiding overfitting hyperparameters.

Statement of Contributions

This project was realized by Thierry Jean for the course IFT 6390 during the Fall semester of 2021. I hereby state that all the work presented in this report is that of the author.

References

- Pedregosa et al., (2011). Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830.
- Chen, T. & Guestrin, C., (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794.