Report

Project Title: Classification of extreme weather events Kaggle I competition for IFT3395-6390 Fall 2023 course at Université de Montréal

Full Name: Chirag Goel
Student Number: 20273174
Kaggle Username: chirag_goyal24

11/6/2023

1 Introduction

The Kaggle competition aims to build a machine learning model for the classification of extreme weather conditions based on time point and location, latitude and longitude into 3 classes - Standard background conditions, Tropical cyclone and Atmospheric river. This is is critical for various applications such as disaster preparedness and climate monitoring and is a very challenging task.

The training set contains 44,760 data points from 1996 to 2009, and the test set contains 10,320 data points from 2010 to 2013. Each data point consists of 16 atmospheric variables such as pressure, temperature and humidity, besides the latitude, longitude and time. The data is highly imbalanced with tropical cyclone being the smallest class (in number). Apart from this, the difference in scales of different features also presents a unique challenge.

There were a few machine learning algorithms used for this problem with logistic regression being the major focus. There was some feature engineering done keeping in mind the assumptions of logistic regression and model optimizations performed using the dataset information. Other ML models like SVM and XGBoost were used but the self implemented logistic regression came out on the top.

This report summarises the approach used for feature selection and building the different models throughout the competition and discusses the results.

2 Feature Design

For designing the features there are a lot of methods that one can use including adding transforms, normalisation, removing outliers etc. Square transformations and normalisation was used for the final model building for this project.

When trying outlier removal, a lot of outliers in the boxplot were observed for different features giving the impression that removing these might negatively impact the performance and hence those were not removed. Duplicates were also not removed from the dataset as it negatively impacted the performance.

2.1 Feature Engineering

Logistic Regression assumes that the input features have a linear relationship with the output label which is not always the case. This means that it becomes essential to add non-linearity for getting a good fit for our true distribution. Increasing the degree of the given features using a squared transform is very common for helping the model to capture this non-linear relationship between the features and class labels. This is what was found for this particular dataset too i.e just using the linearity assumption did not give us the best results and hence squared features were added to the feature set.

- **Squared Transform**: To capture the non-linearity, squared features were concatenated with the original features to get a better fit for the logistic regression model.

- **Feature Multiplication**: From 1, negatively correlated features were observed and multiplied together and were further added to the feature set.

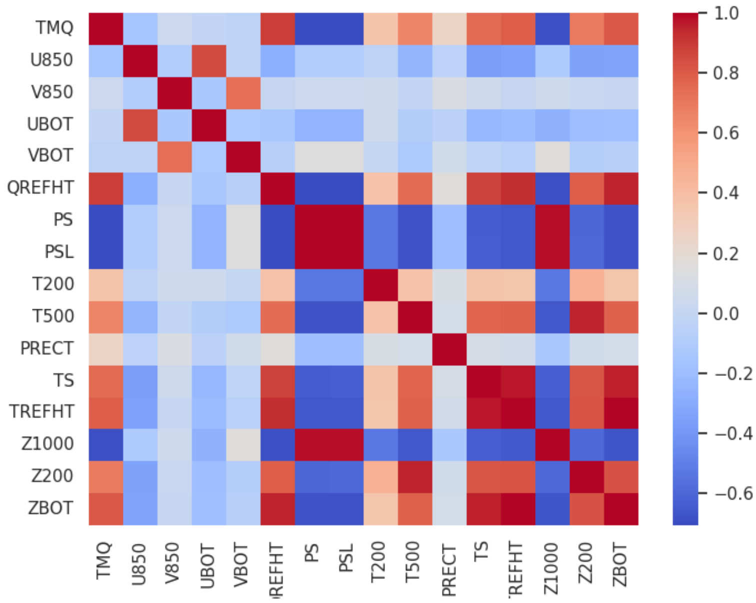


Figure 1: Covariance Matrix Heatmap

This led to the addition of the following features: 'U850_V850', 'UBOT_VBOT', 'PS_PSL', 'T200_T500','Z1000_Z200', 'U850_UBOT', 'V850_VBOT'.

- **Time**: Using the given time column, month was added as an additional feature for just logistic regression which helped improve model performance.

Apart from this other degrees of feature addition was also tried, but the squared transform gave the best results.

2.2 Feature Normalisation

Feature normalisation is another important step of building a ML model. A huge difference in the scale of different features was observed and hence choosing the right feature normalisation becomes important specially for logistic regression. Standard scaler and min-max scaler normalisation techniques were tried and the standard scaler technique worked the best. *Please note that no normalisation is applied for xgboost as it requires the scale of features to build the tree*

3 Algorithms

There were a number of ML models that were used for this competition including k-NN, Naive Bayes, Random Forest, Logistic Regression, SVM, XGboost etc but Logistic Regression, SVM, XGboost are the three that gave us the best results. These will form the focus of our future discussion and can be summarised as follows:

- **Multinomial Logistic Regression**: This is an extension of basic logistic regression algorithm used for binary classification to multi-class classification. This is essential as ours is a 3-class classification problem. Softmax activation and weighted cross-entropy loss were employed for the imbalanced multi-class problem. The loss was optimized using gradient descent with both L1 and L2 regularisation so as to create sparsity among the features because we added additional squared features which might all not be useful and distribute weights uniformly as we added correlated features respectively. This was implemented from scratch in numpy.

The weighted cross-entropy loss is defined as:

$$\text{Weighted Cross-Entropy Loss} = -\sum_{i=1}^{N}\sum_{j=1}^{C} w_j y_{ij} \log(p_{ij})$$

- **SVM**: SVM designed for binary classification for finding the most optimal hyperplane between two classes but can be extended to multiclass classification by using one-vs-one (ovo) or one-vs-rest (ovr) classification. n classifiers are build, according to the type chosen and a class is predicted using all these classifiers. The kernel trick in SVM can be used to find non-linear relationships between the features and the output variable i.e learn a non-linear decision boundary. Hyperparameters like C, gamma, class_weights were tuned for rbf, polynomial and linear kernels. The sklearn library (sklearn.svm.SVC) was used to train the SVM model.

- **XGboost**: XGBoost is a robust and efficient algorithm for multi-class classification, offering flexibility, interpretability, and scalability. Its ability to handle missing data, feature importance analysis, and optimization capabilities make it a popular choice for a wide range of machine learning applications. It is an ensemble learning algorithm based on the gradient boosting framework. It builds a strong predictive model by combining the predictions of multiple weak learners, typically decision trees.

4 Methodology

After the addition of features, the training data was split into a training and validation fold with every $5th$ index in the original data ending up in validation, which is a (80%, 20%), (train, val) split. This was used for hyperparameter tuning and for the final results on the test set the entire training dataset (train+val) was used.

4.1 Multinomial Logistic Regression

This algorithm was the major focus of the competition. The loss function (weighted cross entropy) was optimized using gradient descent optimizer.

Regularizer: The optimization was aided with the use of elasticnet regularization i.e a sum of both L1 and L2 loss penalties. This adds sparsity to the weights so as to disregard the irrelevant features which might have been introduced as a result of our feature engineering. This also helps in handling highly correlated features, improves robustness, and hence improves performance.

Weighted Loss: The dataset was highly imbalanced and just using regularisation did not solve our purpose. To handle this imbalance, the loss function was penalised according to the weights of class. This was calculated in two ways:

$$weight\_class_i = \frac{|y \neq \text{class}_i|}{|y|} \tag{1}$$

$$weight\_class_i = \frac{1.0}{|y = \text{class}_i|} \tag{2}$$

Out of the two equations 1 performed the best and was used for final submission.

Hyperparameter Tuning: The hyparparams- learning rate, lambda1, lambda2 - were tuned using an hyperparameter list of $[0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.5, 0.9]$ where lambda1 is the learning parameter for L1 regularizer, lambda2 is the learning parameter for L2 regularizer and learning rate is the parameter for gradient descent. This was done using the (train, val) split created and the best hyperparameters $num\_epochs : 3000, learning\_rate : 0.1, lambda1 : 0.001, lambda2 : 0.001$ were chosen.

The best model was chosen based on the accuracy for any number of epochs run.

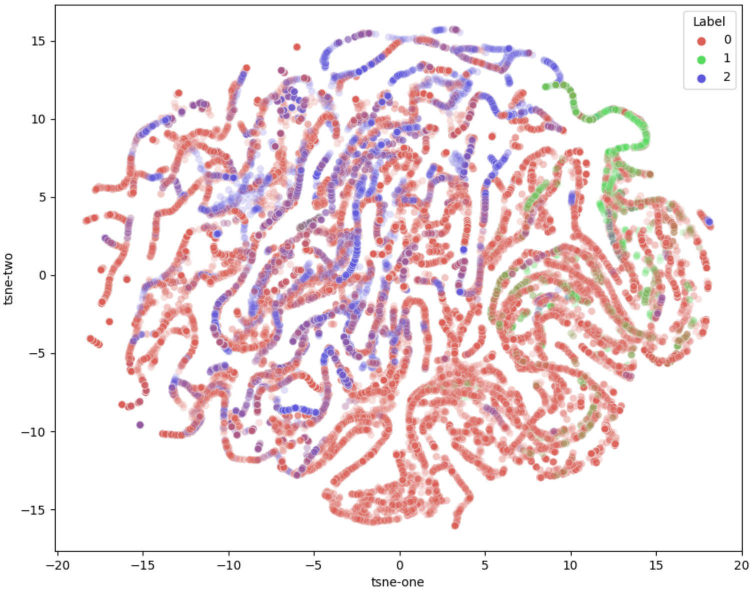


Figure 2: 2D TSNE plot for the given dataset without any additional features

From fig 2, we can clearly see that that the data distribution is not linear and hence we require a non-linear classifier for classification. Even though we tried to add the non-linearity by feature engineering but that might not be good enough. So, it is wise to move towards non-linear models like SVM or XGboost. *Please note that this observation should be interpreted carefully as it is possible that in higher dimensions we can find a linearly separable hyperplane.*

4.2 SVM

The sklearn implementation sklearn.svm.SVC was used for SVM training. The hyperparameters available for tuning are: kernel, C, gamma, class_weight. RBF, polynomial and linear kernels were used on both the raw dataset and the dataset with additional features for different combinations of C, gamma and class_weight by using the same (train, val) dataset split. The type of classifier was fixed to one-vs-one.

The usage of class_weight led to far worst results and hence it was set to None. Now,to cater to the imbalance in dataset, SMOTE was used to oversample the dataset by creating small variations of classes with lower frequency. But this again led to a worst model and hence was dropped.

Kernel, C and gamma were optimized using the (train,val) split and polynomial kernel with $C = 10$ and $gamma =' scale'$ produced the most optimal results.

4.3 XGboost

The xgboost library was used for model training, using sklearn's k-fold cross validation for hyperparameter optimisation. The features were not normalised as xgboost uses the scale of these features. The objective function used is multiclass softmax loss and the hyperparameters learning_rate, n_estimators, max_depth, min_child_weight, gamma, subsample,colsample_bytree were sequentially tuned to find the optimal combination as learning_rate = 0.05, n_estimators=10000, max_depth=8, min_child_weight=2, gamma=0.0, subsample=0.6, colsample_bytree=0.55, reg_alpha=200, reg_lambda=100.

5 Results

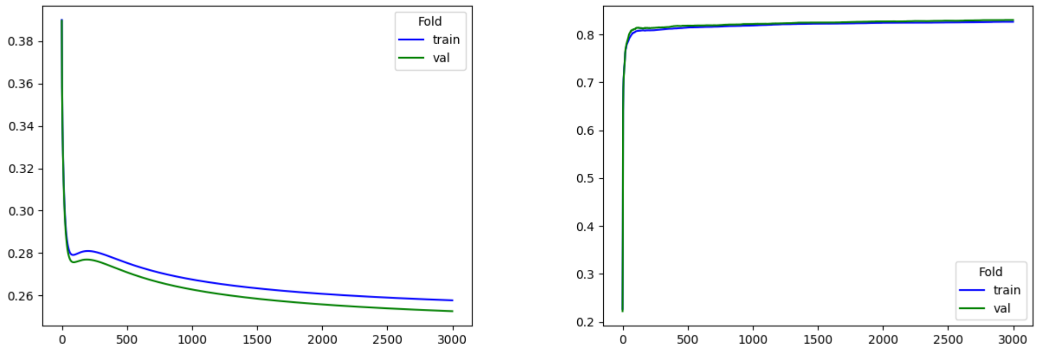
Present a comprehensive analysis of your results, including tables, graphs, and other visual aids as appropriate. Compare the performance of at least two different methods/models. Include a discussion of the most important hyper-parameters and their impact on the results. If you have Kaggle results, mention them, but don't limit your analysis to them. Include your insights on what worked well and what didn't.

From table 1, we see that all the algorithms give decent performance, but the algorithm with the best result is logistic regression. The training curves for the best performing logistic regression can be found in 3. These show that the training converged as we see the training and validation curves follow a similar pattern.

Table 1: Kaggle Results for Models trained with optimal Hyperparameters

Model	**Public Score**	**Private Score**	**Average Score**
Logistic Regression	0.786	0.787	0.7865
SVM	0.749	0.761	0.755
XGboost	0.782	0.773	0.7775

It is the combination of all the hyperparameters rather than any single hyperparameter that causes the performance improvement for the logistic regression classifier which is evident from 2.



(a) Loss

(b) Accuracy

Figure 3: Training Curves for Logistic Regression Training on the (train, val) split

Table 2: Kaggle Results for Logistic Regression model trained with different Hyperparameters

Learning Rate	Lambda1	Lambda2	Average Score
0.9	0.00	0.01	0.7815
0.5	0.001	0.01	0.784
0.1	0.001	0.001	0.7865

6 Discussion

Logistic Regression is performed the best, which shows that simpler algorithms can perform much better than complex algorithms if enough time and effort are spent on them. Since logistic regression is a high bias and low variance model I did not observe a change in my scores for both private and public leaderboard but there was a difference for both SVM and XGboost. This might be considered as both a good and bad thing but in real world, models that perform better on average on the two test sets would perform better.

The feature engineering and hyperparameter tuning were the major part of this project. I believe, the improvements in logistic regression were saturated by just using the same setting. More complex optimizers like Adam can be used which can improve the performance even more. Domain knowledge and better outlier detection algorithm could have also helped me build a better model.

For the non-linear models like SVM and XGboost, I think that the hyperparameter optimization was the biggest challenge, especially for xgboost. I could observe that there were some features that have low feature importance and removing them could have been helpful not just for xgboost but for other algorithms too.

7 Statement of Contributions

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

References

[1] Logistic Regression:https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[2] Implement Logistic Regression: https://rickwierenga.com/blog/ml-fundamentals/softmax.html

[3] L1 and L2: https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261

[4] SVM: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[5] XGboost Hyperparameter Tuning: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[6] XGboost Sample Weight: https://stackoverflow.com/questions/67868420/xgboost-for-multiclassification-and-imbalanced-data