Classification of extreme weather events

Seyed Armin Hosseini[1]

[1]**Student Number: 20257663**
[1]**Kaggle Username: arminhsn**

INTRODUCTION

This project is about the detection of extreme weather events from atmospheric data. The goal is to automatically classify a set of climate variables into three classes: standard conditions, tropical cyclones, and atmospheric rivers. For this project, I implemented multinomial logistic regression from scratch with a momentum-based gradient descent optimizer, RMSprop weight update equation, cross-entropy loss function, and softmax activation function to classify the data. I also performed other machine learning classifiers, including random forest classifier, Naive Bayes, and support vector machines with the scikit-learn library. In logistic regression, I could achieve an accuracy of **78.2%**; **76%** for SVM; **75.5%** for random forest classifier; while for Naive Bayes the results were not good enough to report.

FEATURE DESIGN

For this project, I have done three main prepossessing of the data. First, I changed the space of the time feature. By exploring the data, I found out that we have the feature time, which is not a good feature for our purpose, because its type is string, so it does not add any value to our model and I have to convert it to numerical value by converting it to datetime. Besides the problem of its type, the nature of the feature time does not provide enough intuition for the models about the difference or similarity of the examples because December 29th is near January 1st, so the weather conditions would be similar, but these two dates have different years, and if we just keep the feature time as it is, we lose this important information. A good way to retain this information is to transform the feature time into a sine and cosine space that transforms the data into a circle-like dimension. In this new space, the last day of each year is near the first day of the next year, and also for other days of the year, if the days are similar, their sine and cosine values would be so. Second, the difference between the values in our data is significantly high, and this situation leads to a problem that the model should assign relatively different weights to our features to take care of this situation. But, our weights in the first place are randomly generated with a uniform distribution so achieving this state is very hard for the model. Therefore, we try to avoid having this kind of problem and we normalize our data in the first place. I used min-max normalization because it scales all the data between 0 and 1 and I haven't used standardization because by doing that the problem can still be there. In addition, I have added the powers of two of the features. This method is used to add some non-linearity to the model which by its nature is linear. This feature engineering method is advised by (Brownlee, 2020), especially for logistic and linear regression problems. I can think of this as applying a non-linear transform. Because we have studied in this course we can transform our feature space to a non-linear space in which a hyperplane decision boundary is a non-linear decision boundary in our original feature space. This method is a simple variation of non-linear mapping as it only adds one extra set of features which are powers of two of the original features. There might be a question as to why I didn't use other degrees with more complexity. Because with this transform I got a good result and I didn't want to overfit on data.

ALGORITHMS

As we instructed I have implemented multinomial logistic regression from scratch and I have used support vector machines and random forest from the scikit-learn library.
Here I am going to explain what I have done in logistic regression to improve my accuracy.
First I used vanilla gradient descent which gave me poor accuracy on the test set. Therefore, I decided to use a more advanced version of gradient descent.
In class, we were told that there are some other versions of gradient descent such as momentum gradient

descent. This approach helps accelerate the convergence of gradient descent by adding a momentum term to the weight update. The momentum term is based on the previous weight update and helps the algorithm build momentum as it descends the loss function Jeeva (2023). On the other hand, for a sparse input feature, a large weight update will happen only when the input changes from 0 to 1 or vice versa. Whereas a dense feature will receive more updates. Therefore, using a constant and the same learning rate for all the features is not a good idea. From above we can deduce that the learning rate for a feature should be decayed such that it is inversely proportional to the weight update frequency of that feature. The new learning rate should be low for frequent features whereas it should be high for sparse features. In this case, there is a solution called AdaGrad (Adaptive Gradient). The new learning rate for AdaGrad decays by a factor of the squared sum of the past gradients after each iteration. Although it solves our problem of updating the sparse features, it at the same time introduces a new one. For dense features, the past gradients will be non-zero, and after some iterations, the learning rate shrinks too rapidly due to the accumulation of all past squared gradients in its denominator. To address this issue, we can learn from Momentum-based Gradient Descent. Instead of using all the past gradients in equal proportions, we will use an Exponential Moving Average of the past squared gradients to essentially restrict the window of accumulated gradients to only a few recent ones.
So the new proxy for update frequency is as follows:

$$v(t) = \beta \times v(t-1) + (1-\beta) \times \delta^2(t)$$

$$\Delta w(t) = -\frac{\eta}{\sqrt{v(t)+\varepsilon}} \times \delta(t)$$

The above equation is the weight update equation for RMSprop (Bhat, 2020).
I also used these references to implement this method (Shankhar, 2020) (Brownlee, 2021).

METHODOLOGY

For the choice of training/validation split, I used all the data after the year 2005 for the validation set and the year 2005 itself and before that for the training set which gives a percentage of 18% for the validation set and 82% for the training set. I used this method because the test set time is after the data we were given to work with and I thought it might be a good idea to split my data exactly like what we have as the test set. In other words, the weather conditions might change over the years as a result of the global environmental situation. Therefore, if we have a test set that is for future years of the training/validation set in order to have a good approximation on the validation set we have to make the conditions equal to the test set.
As we were told in class for the multiclass classification with logistic regression with can use cross entropy as our loss function which is the below function.

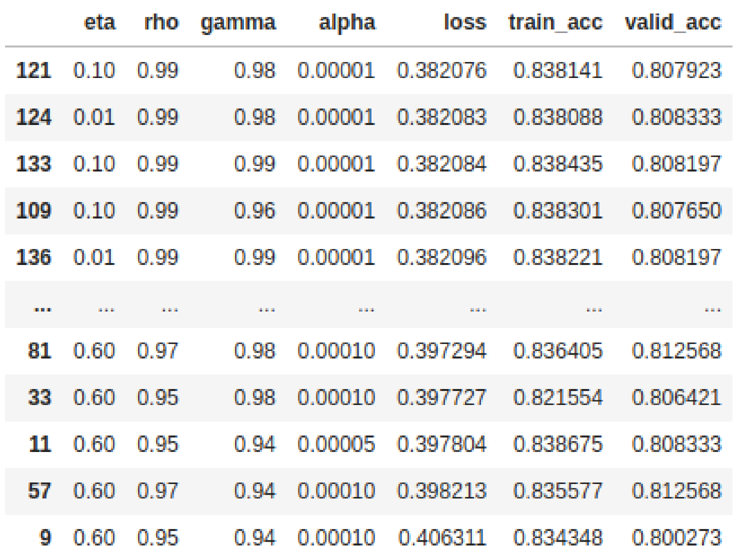
$$L(\mathbf{w}, b) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log p_{ik} + \frac{\alpha}{2}\left(|\mathbf{w}|^2 + b^2\right)$$

This equation combines the cross entropy loss function for multiclass classification with the L2 penalty term for regularization. The cross-entropy loss measures the discrepancy between the true and predicted probabilities of the classes, while the L2 penalty term shrinks the weights and bias to prevent over-fitting. The regularization coefficient α controls the trade-off between the loss and the penalty (Martin, 2000) (Sourabh, 2022).
For the choice of hyper-parameters, first I have to explain what hyper-parameters I have in my logistic regression. I have 4 hyper-parameters which are defined as follows:
η: This is the learning rate
α: This is the regularization term
ρ: This is the momentum hyper-parameter
γ: This is the decay rate hyper-parameter
I have used bellow values for the hyper-parameter tuning:
η = [0.1, 0.01, 0.3, 0.6]
α = [0.0001, 0.00001, 0.00005]
ρ = [0.95, 0.97, 0.99]

γ = [0.94, 0.96, 0.98, 0.99]
I chose α among very small values because in the gradient descent I calculated weights and biases on the mean of the difference between predictions and actual values. Therefore, this value would be very small so I have to choose my regularization term accordingly. In the contrast, my initial learning rate (because it will be changed adaptively) should be a bigger number so that it starts with big steps and adaptive when moving onto the minimum. ρ and γ mostly are chosen in the range of 0.90 to 0.99.

RESULTS

For logistic regression, I applied all the combinations of these hyper-parameters and my results are shown in figure 1.



	eta	rho	gamma	alpha	loss	train_acc	valid_acc
121	0.10	0.99	0.98	0.00001	0.382076	0.838141	0.807923
124	0.01	0.99	0.98	0.00001	0.382083	0.838088	0.808333
133	0.10	0.99	0.99	0.00001	0.382084	0.838435	0.808197
109	0.10	0.99	0.96	0.00001	0.382086	0.838301	0.807650
136	0.01	0.99	0.99	0.00001	0.382096	0.838221	0.808197
...	...	...	...	...	...	...	...
81	0.60	0.97	0.98	0.00010	0.397294	0.836405	0.812568
33	0.60	0.95	0.98	0.00010	0.397727	0.821554	0.806421
11	0.60	0.95	0.94	0.00005	0.397804	0.838675	0.808333
57	0.60	0.97	0.94	0.00010	0.398213	0.835577	0.812568
9	0.60	0.95	0.94	0.00010	0.406311	0.834348	0.800273

Figure 1. Results of hyper-parameter tuning for logistic regression

For the best choice of hyper-parameters my error curve is shown in figure 2 and the accuracy curve is shown in figure 3.

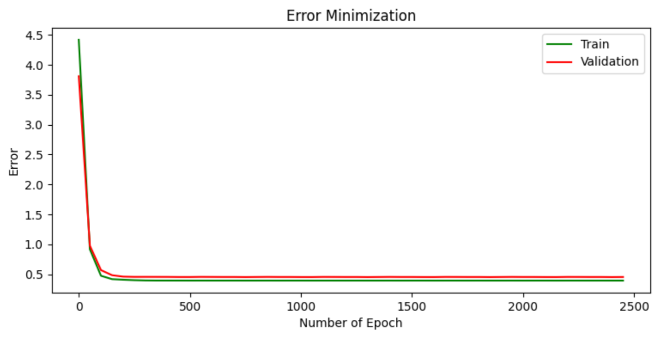


Figure 2. Error minimization curve for training and validation

For the SVM model I tried RBF kernel with bellow values for γ and c:
C = [0.1, 1, 10, 100, 1000]
γ = [0.1, 0.01, 0.001, 0.0001]
And polynomial kernel with [1, 2, 3, 4, 5] degrees which the five best results are shown in table 1. And for the random forest I used bellow hyper-parameters:

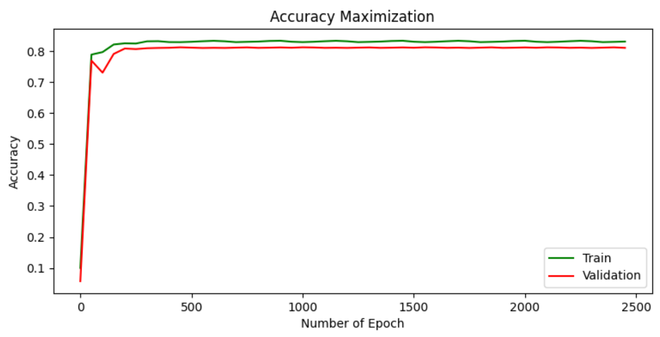


Figure 3. Accuracy maximization curve for training and validation

Kernel	C	γ	degree	Train accuracy	Validation accuracy
polynomial	0	0	2	0.854	0.812
rbf	100	0.01	0	0.845	0.810
rbf	1	0.1	0	0.840	0.809
polynomial	0	0	3	0.868	0.809
rbf	100	0.1	0	0.873	0.807

Table 1. Results of hyper-parameter tuning for SVM

n_estimators = [100, 200, 300, 400, 500]
max_depths = [5, 10, 15, 20, 25, 30]
min_samples_splits = [2, 5, 10, 15, 20]
min_samples_leafs = [1, 2, 5, 10, 15]
The five best results are shown in table 2

n_estimators	max_depth	min_samples_split	min_samples_leafs	Train accuracy	Validation accuracy
200	5	15	1	0.836	0.818
200	5	15	2	0.836	0.818
200	5	15	5	0.836	0.818
200	5	10	1	0.836	0.818
200	5	5	1	0.836	0.818

Table 2. Results of hyper-parameter tuning for random forest

The best model among these three models based on the accuracy on the test set is the logistic regression. and as we can see in the results the most important hyper-parameter for the logistic regression in the α which is the regularization term. And in the SVM model kernel has the most importance and in RBF kernel c and γ are mutually contribute to the accuracy of the model and in random forest max_depth and n_estimators are the most important hyper-parameters.

DISCUSSION

I have used advanced versions of gradient descent which could be the reason that it worked well on this data in comparison with SVM and random forest. The problem here is that in enterprise machine learning solution they always use tools in order to store their model and choice of hyper-parameters such as: MLflow and comet.ml which make hyper-parameter tuning easier and help with reproducibility. In addition, I could have used some other methods in the scikit-learn library such as GridSearchCV and pipeline to make the hyper-parameter tuning process faster and more accurate.

STATEMENT OF CONTRIBUTIONS

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

REFERENCES

Bhat, R. (2020). Adaptive learning rate: Adagrad and rmsprop. https://towardsdatascience.com/adaptive-learning-rate-adagrad-and-rmsprop-46a7d547d244//.

Brownlee, J. (2020). How to use polynomial feature transforms for machine learning. https://machinelearningmastery.com/polynomial-features-transforms-for-machine-learning/.

Brownlee, J. (2021). Gradient descent with momentum from scratch. https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch//.

Jeeva, C. (2023). Momentum-based gradient descent. https://www.scaler.com/topics/momentum-based-gradient-descent//.

Martin, D. J. . J. H. (2000). *Speech and Language Processing*. Pearson.

Shankhar, B. S. (2020). Rmsprop. https://medium.com/optimization-algorithms-for-deep-neural-networks/rmsprop-fb992098fa6e//.

Sourabh (2022). How to build a robust logistic regression model with l2 regularization? https://analyticsindiamag.com/how-to-build-a-robust-logistic-regression-model-with-l2-regularization///.