IFT6390 Kaggle Competition 1 Report

Vitaly Kondulukov
Kaggle Username: kondvit
Matricule: 20198956

November 2021

1 Introduction

This report is about the results of weather events classification competition on Kaggle in the context of IFT6390 - Fundamentals Of Machine Learning class at University Of Montreal. The training set is composed of 47,760 labeled data points, from 1996 to 2010, and consists of 16 atmospheric variables. The data points are classified into one of three classes: standard background conditions, tropical cyclone, atmospheric river. The test set contains 7,320 data points, from 2011 to 2013. Logistic Regression model was implemented from scratch and trained on the training set. With minimal feature engineering the Logistic Regression model yielded 75% classification accuracy on the held out test set. Scikit learn's random forest classifier yielded the best accuracy on the validation set and the best final accuracy on the test set, which is 77.814%. In case of Random Forests, the training data was balanced out by SMOTE oversampling the unbalanced classes.

2 Feature Design

The original training data set contained 21,997 duplicates, the same features and labels. Those were removed. The time column was in an integer formal: yyyymmdd. This format was broken down into two features year (yyyy) and month/day (mmdd). The month and day was kept together since it formed a natural ordering from 0101 (1st January) to 1231 (31st December). This was done to indicate how late in the year/month were the examples recorded. The new processed data set still had some problems. There were around 6k duplicated data points where the features were the same but not the labels. For example, we could have two data points labeled differently, but with the same features. Two attempts were made to fix this problem. First attempt was to exclude those uncertain data points all together. This led to overfitting on the training set and the validation set. The pruned random forests model yielded 99% on the training set and 98% on the validation set, but only 69% on the Kaggle test set. The second attempt was to drop the labels and to keep out of the 6k duplicates only unique data points. Then we would classify them with KNN (k=5) that was trained on the data set with no duplicates at all. This also led to overfitting, the validation set yielded 98% accuracy and the Kaggle test set, 74%. Therefore, only the duplicates of features and labels together were removed from the original data set to train our final models. The classes 1 (Tropical cyclone) and 2 (Atmospheric river) were severely unbalanced respectively: 19295, 1218, 5250. SMOTE oversampling technique was used to balance the dataset after duplicates removal.

3 Algorithms

Two algorithms were used for this classification problem: Logistic Regression and Random Forest Classifier.

3.1 Logistic Regression

Logistic Regression attempts to minimize empirical risk of the data through a surrogate log likelihood loss using gradient descent. Three separate models were trained using one-vs-all paradigm. Three separate label sets were created, one for each class. In the respective set, a class of interest would be labeled as 1 and all the other data points as 0. Each model, given a fresh example, would predict a probability of it's respective class. The predictions would then be based on the highest probability outputted among the three models. It's respective class would then be predicted.

3.2 Random Forest Classifier

Random Forest Classifier as described by the user guide on the sklearn website: "A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting". The hyperparameter ccp_alpha was used to prune the trees and to control the overfitting.

4 Methodology

For both models, the performance evaluation and hyperparameter adjustments were done on a 80%/20% training/validation set split with random shuffling for each training. Various models were trained and validated before settling on random forests. The confusion matrix as well as precision and recall were used to settle down on the random forests model as the final one. The final models that were used on the test set were

trained on the whole training set without validation split.

4.1 Logistic Regression

Logistic Regression required us to choose three hyper parameters: learning rate, regularization coefficient and number of iterations for the learning loop. The search was done in basic greedy way. Two out of three parameters remained fixed and the third one was adjusted and trained in a loop. Then, the results of accuracy on the validation set of each iteration was used to compare each value of the hyperparameter.

4.2 Random Forest Classifier

The only hyperparameter that was adjusted from the default sklearn's parameters is ccp_alpha. Same greedy strategy was used for the ccp_alpha. Starting with 0 and incriminating by 0.00001 in each loop, 100 times. Likewise, in every iteration the model was trained and evaluated on a fresh train/val set split. This is to prevent overfitting on the validation set. K-Fold validation would take too long to execute, this approach is a nice balance between time and integrity of the parameter search. The model with the highest accuracy on the validation set corresponded to our final choice of ccp_alpha.

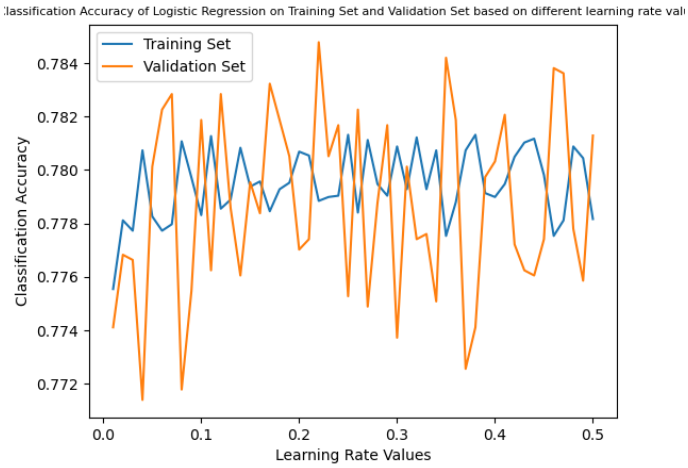
5 Results

5.1 Logistic Regression

A greedy search was performed for each of the three main hyperparameters of our logistic regression model: learning rate, ridge regularization weight and number of iterations of the learn loop.

5.1.1 Learning Rate

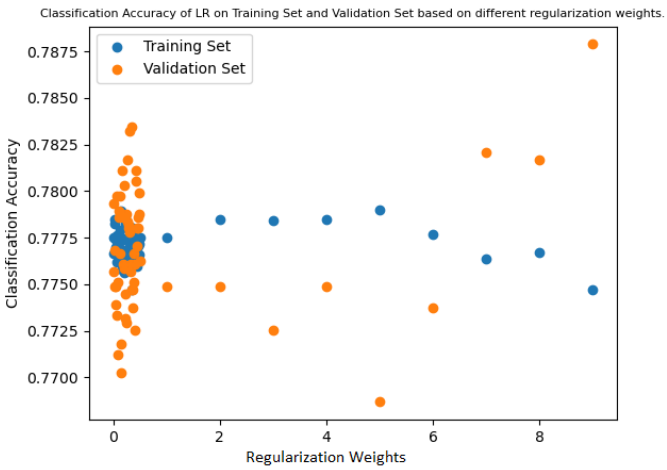
We have tried multiple learning rates from 0.01 to 0.5 and plotted validation/training set accuracy percentages versus those values. Ridge regularization weight was set to 0 and the number of iterations was set to 1000 during all of runs.



As we can observe, all of these learning rates perform similarly. Both of the line plots look sporadic at first glance, but if we take a look at the scale of the y-axis, our validation/training accuracy is within 1%. For our final model we chose a middle ground of 0.2 for learning rate.

5.1.2 Regularization Weight

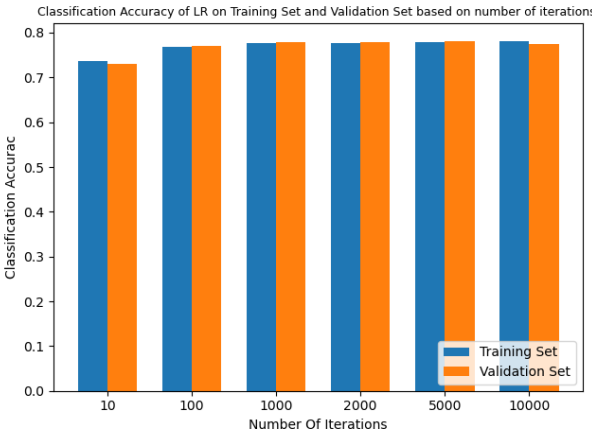
For the ridge regularization weights, we tried weights from 0.01 to 0.5 in increments of 0.01 and from 1 to 10 in increments of 1. This covered a wide spread of weights. The following scatter plot shows the results.



As we can observe, the regularization weights did not play a large role in the model accuracy. Therefore, for our final training, we chose not to use regularization.

5.1.3 Number Of Iterations

For the number of iterations, we tested six numbers: 10, 100, 1000, 2000, 5000 and 10000. In all cases, the learning rate as 0.2 and regularization weight set to 0. We have obtained the following accuracy scores on training and validation sets:

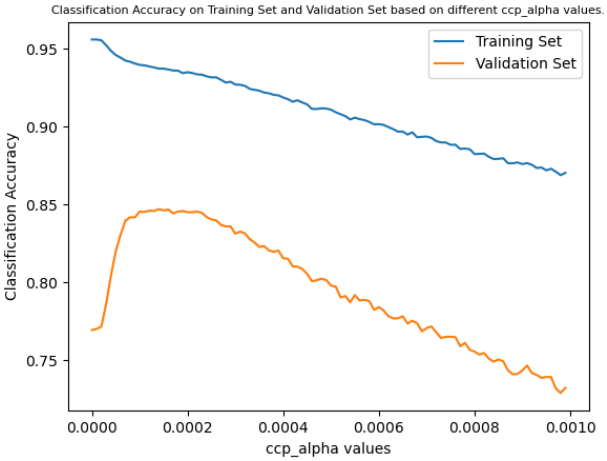


As we can observe, from 1000 iterations the algorithm seems to converge to a local optimum. Therefore, we chose a 1000 iterations for our final model.

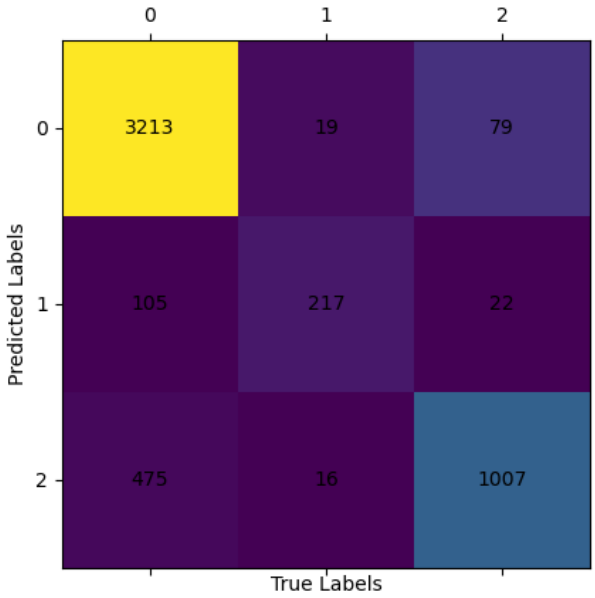
Our final logistic regression model was trained on the whole pre-processed training set with learning rate 0.2, regularization weight 0 and number of iterations 1000. We obtained 78.10% accuracy on the validation set and 75.00% accuracy on the Kaggle test set.

5.2 Random Forest Classifier

As discussed in the Methodology section, a greedy search was done over different ccp_alpha values starting from 0 in increments of 0.00001 until 0.001. This hyperparameter as quoted from the sklearn's documentation is: "Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than ccp_alpha will be chosen." The rest of the hyperparameters were left at default setting. Notably, the number of trees remained 100. The following graph demonstrates the relation between the accuracy on training/validation sets and different ccp_alpha values.



As we can observe, around ccp_alpha = 0.0001 is a good choice that maximizes validation set accuracy. We take a hit on the training set accuracy, but that is to be expected. We don't want to overfit, that is why we are taking this tradeoff. The two line plots are jagged because we are shuffling the data and using a fresh train/validation set split for every alpha, there is an effect of randomness. For our final model, we trained RandomForestClassifier on the whole pre-processed training set with ccp_alpha = 0.0001. We obtained 86.10% accuracy on the validation set and 77.81% accuracy on the Kaggle test set. The following plot shows the confusion matrix.



Our precision of classes 1 and 2 suffers and is something that desires improvement.

6 Conclusion

The methodology for the choice of hyperparameters was discussed under Methodology section. For each new value of a hyperparameter, we reshuffled the data and made a train/validation split. This is not the best framework to use. Idealy we would like to use k-fold cross validation on the dataset for each new value of a hyperparameter. However, the method we used provides a balance between computation time and integrity of the findings. Since we create a new validation set everytime, we avoid overfitting on any particular validation set. Moreover, both of our models suffer from poor precision of the classes 1 and 2. Some balancing techniques or different classification thresholds during predictions might improve precision for those classes.

6.1 Statement of Contributions

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

7 References

Decision Trees:
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

RandomForestClassifier:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

SMOTE:
https://machinelearningmastery.com/smote-oversampling-
for-imbalanced-classification/