EURECOM
Sophia Antipolis

AML Spring 2022

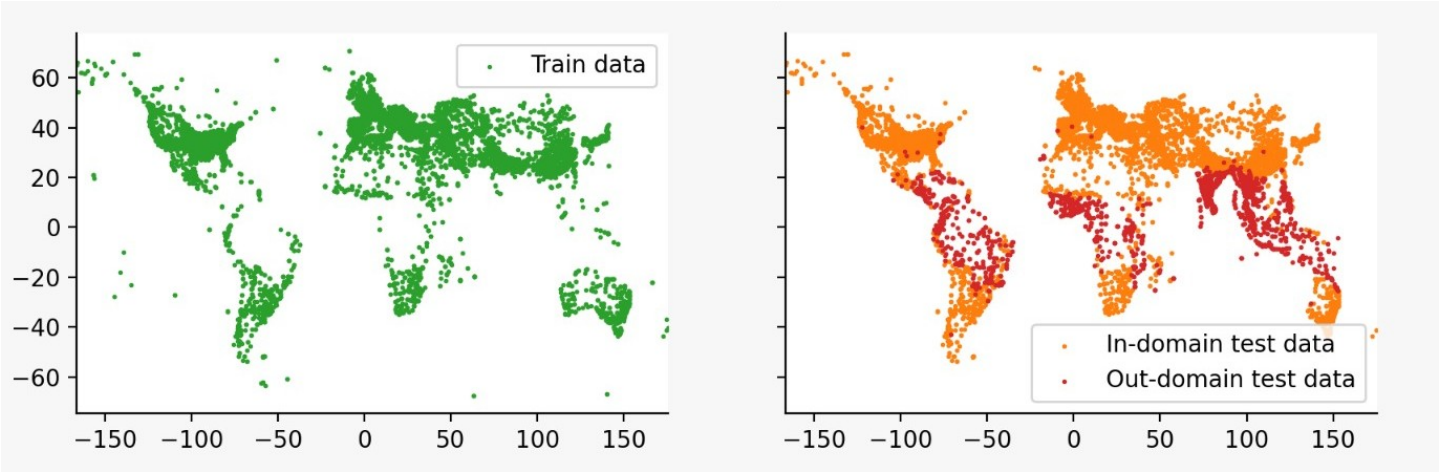
Challenge 1: Weather prediction

September 19, 2022

Group 8: Harkati Chiraz Rayene, Senane Zineb, Zheng Estelle

Emails: {Chiraz.Harkati, Zineb.Senane, Estelle.Zheng}@eurecom.fr

Professor: Michiardi Pietro

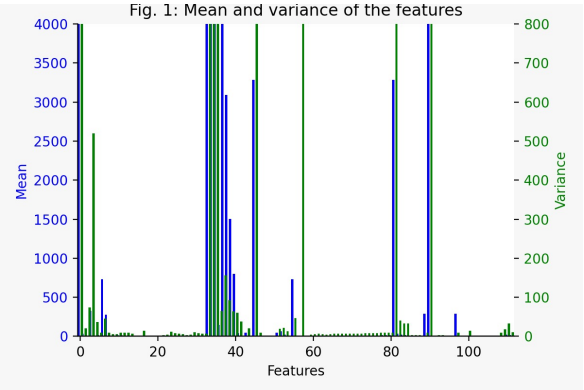


1 Introduction

This challenge aims to build models to predict the temperature at different locations around the globe (Dry, Mild and Tropical areas) given measurements in Dry and Mild areas as a training dataset. It is a regression problem where we are faced with the concept of covariate shift: predict labels of data that are not in the domain of the training. For that purpose, we analyzed the data and preprocessed it to clean the data and decide which features to consider. Then, we trained and compared different machine learning models to select the best one in terms of root-mean-square error (RMSE).

2 Data Analysis & Preprocessing

The training dataset contains 1,993,574 records which are pairs of 111 meteorological numerical features and target values (the temperature at a particular latitude/longitude and time). There are no duplicate records but 108,848 of them contain one or more features with missing values (1,154,176 in total). As shown in Fig.1, the features are highly heterogeneous (i.e. they are of different types and scales) and some of them have very small variances (i.e. close to 0) and these are outliers. Because the dataset is very big with numerous features, reducing the input space might be useful and even crucial for computation efficiency. Moreover, the correlation matrix of the features reveals that some are negligibly correlated with the target which motivates the reduction of the input space.



2.1 Data Preprocessing

2.1.1 Handling outliers

Since the variances of 20 features are negligible (variance < 0.5), we dropped these outliers as they cannot be used to find any interesting patterns and distinguish different data, which gives us 91 remaining features. Removing these outliers does not affect our prediction results, since they are considered separated and detached from the given data, and can even improve the prediction.

2.1.2 Handling missing values

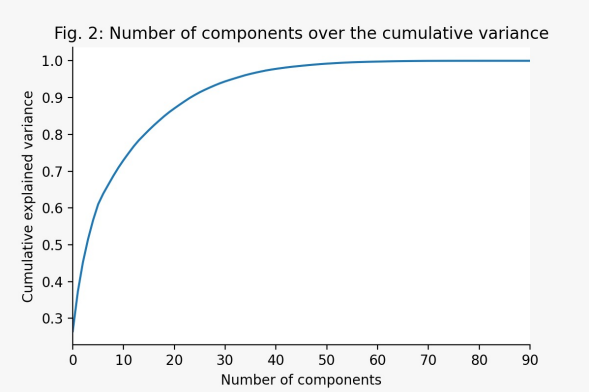
After removing outliers, there are still 108,848 records with missing values, for a total of 814,767 missing values. A naive possibility would be to delete these records, which would reduce the size of the training dataset to 1,884,706 records, but it is still a large dataset.
However, these records can be preserved through data imputation: replacing missing values with the mean of the corresponding column (since all features are numerical).
We compared the results by performing all the preprocessing with and without imputation, and we obtained slightly better predictions without imputation. This is because most samples have missing values for several of the same features, and replacing these missing values may change the distribution and bias the model. Therefore, we did not use any data imputation subsequently.

2.1.3 Handling heterogeneous data: Standardization

Due to the high heterogeneity of features, scaling data is needed to bring all features in the same standing when the scale of a feature is irrelevant or misleading, for example, inputs for neural networks or PCA (Principal Component Analysis). Indeed, the model could be biased as big scaled features become dominating. However, standardization is not always necessary (in random forests for instance) and should not be performed when the scale is meaningful. As we will perform PCA just after this step, we standardized the data.

2.1.4 Reducing the input space

Processing 91 features with large datasets can take a huge amount of time and space depending on the complexity of the model. Hence, we can reduce the input space to help the computation efficiency by dimensionality reduction or feature selection.



PCA performs dimensionality reduction by transforming the training data into a lower-dimensional space. Each principal component (PC) is a linear combination of the original features and is uncorrelated to others. They are ordered such that the first few ones explain most of the variation of the original features. As shown in Fig. 2, by keeping 30 PCs, more than 90% of the variance is retained, and taking more PCs does not increase the explained variance considerably. Thus, we chose 30 PCs to perform PCA.

We also performed feature selection in parallel, by selecting the 30 features that have the best scores for the mutual information metric, which reduces the number of features by 3. Before that, we first sampled 100,000 samples randomly because computing the mutual information with the whole dataset would have taken too much time. However, when comparing the errors for predictions using the Lasso linear model with either PCA or feature selection, feature selection reduced notably the results. This is probably because we discarded some relevant features and considering more features might have improved the performance. But not knowing the number of features to keep is a drawback of this method and thus, we kept the preprocessing with PCA.

3 Model Selection

To perform our predictions, we introduced linear regression models, random forests, and neural networks. To compare the models, we used the root mean squared error metric.

3.1 Linear Regression

First, we started with a linear approach to model the relation between the features and the labels using linear regression with regularization in order to avoid overfitting. This choice is motivated by the simplicity of this model and its linear time complexity, which is useful to refer to as a baseline. We tried first with a Lasso regularizer (L1 norm) as it tends to make coefficients to absolute zero (sparsity) which can eliminate not relevant features. Then, we also tried with a Ridge regularizer (L2 norm) because compared to Lasso, it does not shrink coefficients to absolute zero and thus works better when most features impact the response. For each of them, we use a grid search to fine-tune the regularization hyperparameter.

3.2 Random Forest Regressor

Because linear models are too naive as our problem is complex and not linear, we thought about using nonlinear models like decision tree regressors. They offer many advantages such as their interpretability (by looking at the splits, we know which features are the most important) or the fact that they do not require any transformation of the features (they can handle missing values and outliers, with no need for scaling, etc). However, decision trees are known to suffer from overfitting problems (even more for problems with a big dataset as one single tree may grow a lot of nodes to separate this huge data) and are highly sensitive to hyperparameters. But most importantly, they have a large variance and are unstable, i.e, a small change in the data can lead to a large change in the structure of the optimal decision tree. A way to reduce variance and overfitting is to use ensemble learning methods like bagging (e.g. random forests) and boosting (e.g. XGBoost). Thus, we decided to directly implement a random forest regressor as it benefits all the advantages of decision trees while reducing variance by training many decision trees on different subsamples of the data and then combining the output of all the trees based on the majority voting. It also works well on large datasets.

We played with some of its hyperparameters such as n_estimators (the number of trees in the forest), max_depth (the maximum depth of a tree), and min_samples_leaf (the minimum number of samples required to be at a leaf node) in order to get better performances. However, because the dataset and the number of features are big, it requires enormous computational power and resources, and the time complexity is also huge, even more, if we fine-tune the hyperparameters. We used some optimization that we will explain later to still be able to use this model, notably, reducing the training set.

3.3 Multi-Layer Perceptron Regressor

MLP is a feed-forward artificial neural network that learns adaptively since, at each step, the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. This allows the model to learn how to do predictions based on the data given for training. They are also well known for their good performance on nonlinear and large data involving many features because they introduce complexity with non-linearity. All these reasons lead us to try this model.
We did not try to fine-tune hyperparameters with a grid search for MLP because it would have taken too much computation time, as we need to keep a large amount of the data for training for MLP. Plus, the MLPRegressor method already has an intern optimizer which is set by default to Adam algorithm. Thus, we chose parameters with prior knowledge (what we think would be good enough to have good performance): three hidden layers of 300, 100, and 100 neurons, learning rate equals 1e-03. We also put an early stopping (if validation loss increases, it stops the training) to avoid overfitting and set the validation set size at 20%. We also set tolerance for the optimization at 1e-03 and 1e-04 with a maximum number of iterations of 30 and 50 respectively.

4 Model Performance

The preprocessing is necessary for MLP as it is unable to handle missing values and requires scaling. However, random forest does not need this preprocessing but we applied it anyway for time and computation efficiency as it reduces the number of features. This preprocessing is only done for finding the best hyperparameters, which we used later on, to train a random forest regressor directly on the unscaled data, without PCA as well.
As we are dealing with large real data known to have high variability and as we are faced with a covariate shift problem, we can keep a bigger portion for the validation set and split 10/90 for train/validation. Indeed, since the dataset is huge, 10% of it still represents hundreds of thousands of records. Having a large validation set can also help the model to reduce overfitting by increasing the model's generalization ability and have good performance even when facing data drift. This is also encouraged in order to reduce the time complexity for training and optimization. However, for MLP, we split 80/20 for train/validation because the model requires a lot of data.
For optimization, we used a grid search with the different sets of hyperparameters for each model, except for MLP regressor as explained above, combined with 5-fold cross-validation to stabilize the best model. After tuning the hyperparameters of each model, we kept the best combination of these parameters and we used it to make the prediction on the whole testing dataset and see if our models generalize and address the data drift issue well.

Model(% train samples)	Train RMSE	Val RMSE	Hyper parameters tuning	Cross Val RMSE	**Test RMSE**
LR with Lasso (80%)	2.640	2.646	alpha=0.001	2.608	**2.371**
LR with Lasso (10%)	2.637	2.640	alpha=0.001	2.603	**2.373**
LR with Ridge (10%)	2.6	2.603	alpha=10	2.603	**2.374**
RF (10%)	3.029	3.212	max_depth=10, min_samples_leaf=10, n_estimators=300	3.165	**2.791**
RF Without Preprocessing (10%)	2.187	2.313	max_depth=10, min_samples_leaf=10, n_estimators=300		**2.140**
MLP (80%)	2.025	2.073	tol=1e-3,max_iter=30		**1.988**
MLP Without PCA (80%)	1.872	1.908	tol=1e-3,max_iter=30		**1.828**
MLP Without PCA (80%)	1.753	1.828	tol=1e-4,max_iter=50		**1.773**

Table 1: Comparison between the different models' performance

For linear models, we tried linear regression with different regularizers and different training set percentages. The first model was trained with 80% of the training dataset and optimized with L1 regularization and it gave a test RMSE of 2.371, as shown in Table 1. We trained the same model but with 10% of the training dataset and the test RMSE was practically the same (2.373). This shows that having 10% or 80% of data for training does not change the model performance much because the underlying model is too simple to explain the whole data distribution and suffers from underfitting. This is highlighted even more with the results obtained for the Ridge linear regression. Even by applying another regularizer to a linear model, we got the same performance (test RMSE being 2.374), which is fundamentally linked to the fact that linear models will underfit in this problem and cannot perform better than this with the same preprocessing.

For random forest, we introduced nonlinear relationships between features to model this complex data. However, the model obtained after preprocessing and optimization does not perform better than simple linear models as we would expect. The test RMSE is 2.791 whereas linear models' ones are 2.37. This is due to the fact that a random forest does not perform well when features are monotonic transformations of other ones, as it makes the trees in the forest less independent of each other. But this is the case when we apply PCA, as each new feature becomes a linear combination of original features, and thus original ones cannot be split properly. Looking at the test RMSE of random forest regressor without preprocessing (test RMSE = 2.14), it confirms that it is better than linear models, but it was PCA that degraded the performance in the first case. The best hyperparameters found are max_depth=10, min_samples_leaf=10 and n_estimators=300. This is reasonable to avoid overfitting since max_depth should not be too high, otherwise trees will grow deeper and have more splits which will be too many for only 30 features after preprocessing. Also, min_samples_leaf should not be too small, otherwise it will increase the depth of each tree and lead to overfitting.

For MLP, we trained on 80% of the training dataset, and we obtained better results than with random forests. This is explained by the fact that with deep enough hidden layers and neurons, added to the intrinsic nonlinearity, we can approximate continuous function more accurately. As we have a large dataset in this problem, MLP will work well as it will benefit fully from the big dataset and learn as much as possible with all this data, which is not the case with the other models we tested. We obtained better performance with MLP without PCA than with it (test RMSE being 1.828 compared to 1.988). This is because PCA is only a transformation of a large set of variables into a smaller one that still contains most of the information but does not retain 100% of the explained variance. Then we fine-tuned the hyperparameters by hand by setting the tolerance for optimization lower (to 1e-04) and allowing a higher maximum for iterations. This results in more iterations in the training of the MLP regressor which further improved the test RMSE: 1.773, which is the best we obtained. By fine-tuning these parameters better, we can obtain even better results.

5 Conclusion & Further Improvements

To conclude, we did some preprocessing to clean the data and reduce the input space for time and computation purposes. But this preprocessing can affect the models' performance. Hence we used them only as a first step for optimizing the models. To handle the data drift problem, we chose to use a large validation set and fine-tune the hyperparameters to avoid overfitting. In this way, the model trained on the training set can be better generalized to test data and perform predictions for unseen samples. MLP without applying PCA was the best regressor that we selected since it gave us the best test RMSE and suits well to problems with a large amount of data. However, a random forest regressor could be another good model in this case, because it automates internally data imputation to fill missing values, and it can select the most important features intrinsically.

To further improve the predictions, an appropriate feature selection can be performed by identifying and removing not only outliers but also drifting features i.e. features that cause the data drift. This can be done by randomly sampling from train and test data and adding a new feature about their "origin" dataset, and then training and testing a model with the "origin" feature just created as the target variable for this model.