Weather Type Classification

Yucheng Ma

School of Computer Science

https://github.com/markfromcd/Weather-Type-Classification.git

1. Introduction

Weather predictions impact a wide range of industries and everyday life decisions. For instance, real-world applicability, safety and disaster preparedness, environmental protection. In detail, accurate weather predictions are crucial for agriculture as farmers plan planting, irrigation, and harvesting schedules to optimize crop growth and minimize risks; in transportation, where airlines, shipping companies, and logistics firms plan routes to ensure safety and efficiency; and in retail, where businesses adjust inventory and marketing strategies based on forecasted weather conditions, such as stocking more umbrellas and raincoats in anticipation of rain. As a result, this is a good and useful problem to be developed and solved.

The dataset contains temperature, humidity, wind speed, precipitation percentage, cloud cover, atmospheric pressure, UV index, season, visibility, and location, the objective is to predict the weather type (e.g., Rainy, Cloudy, Sunny, Snowy). For location, it contains 9 different values, and each value has many corresponding points. **As a result, this is a multiclass non-IID classification problem**, where each weather type is a discrete category.[1]

A highly rated previous work[2] used SVC, decision tree and random forest to build models and train the dataset. The accuracies turned out to be 0.8745, 0.8981, 0.9082. His best model is random forest with no specific parameters. After having my models, the best model score turned out to be 0.9129, which is a little better than previous work.

2. EDA

2.1 Data frame of the dataset:

- There are 13, 200 data points in total, with 11 original columns.
- The target feature is uniformly distributed across 'Rainy', 'Cloudy', 'Sunny', and 'Snowy', each with 3300 instances.
- ‘Weather Type', 'Location', 'season', and 'cloud cover' are categorical features, while the remaining features are continuous.
- When looked deeper, I found that ‘Location’ contains ‘inland0’, ‘inland1’, ‘inland2’, ‘mountain0’, ‘mountain1’, ‘mountain2’, ‘coastal0’, ‘coastal1’, ‘coastal2’ corresponding many points, which means it will be the key to split points into different groups to avoid data leakage.
- There are no missing values.
- I also checked the number of unique values in each feature to determine if any of them are needed to be removed. However, only 'Atmospheric Pressure' has too many values (5456). But since it behaves differently under various weather conditions, it should not be removed. The figure 1 is shown below.

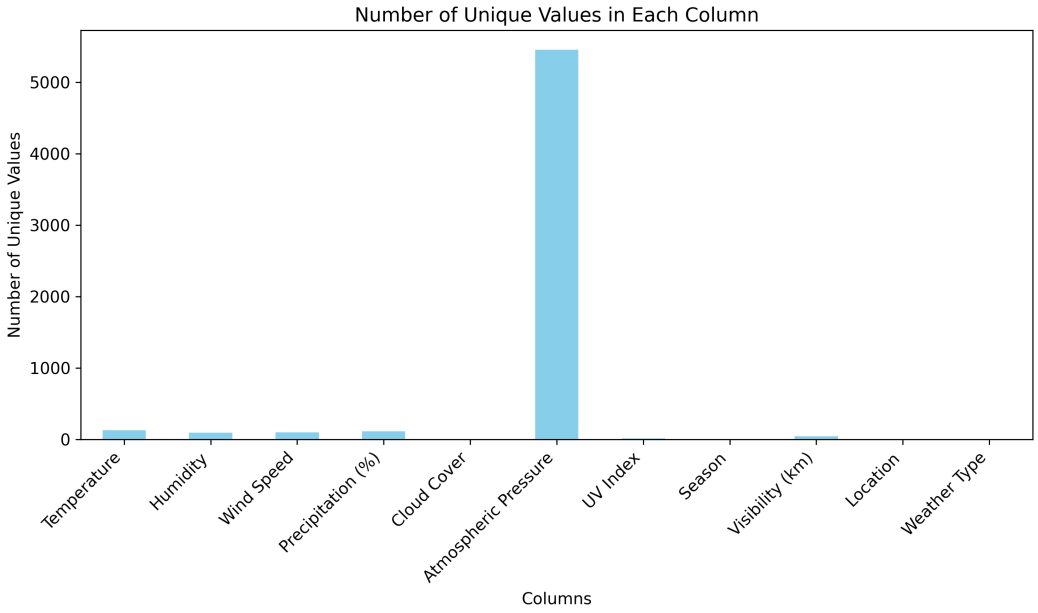

Figure 1: Number of Unique Values in Each Column

2.2 Findings among features:

- The distribution of categorical features is as follows (Figure 2):

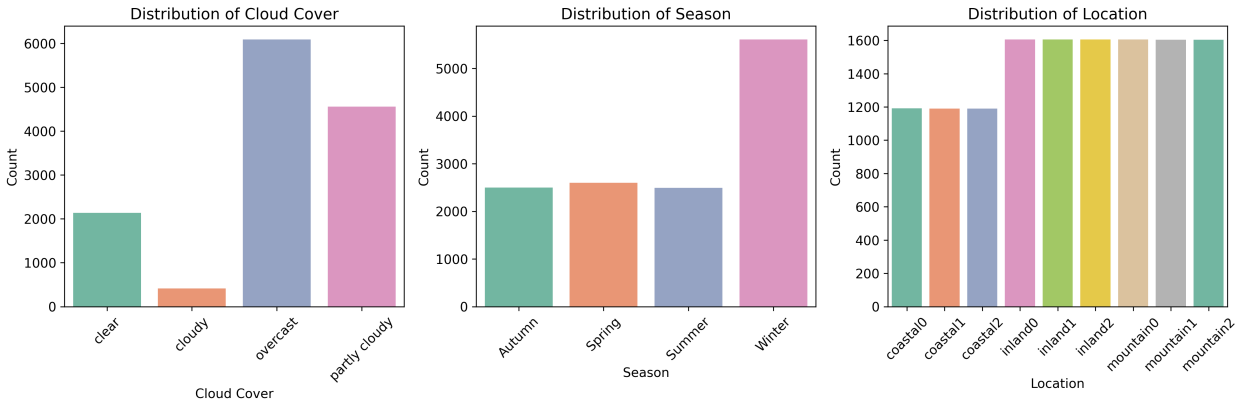

Figure 2: The distribution of categorical features

- Categorical Features vs. Weather Type. Figure 3 illustrates how weather types vary significantly with cloud cover, season, indicating clear, overcast, and partly cloudy skies are predominantly sunny, while seasonal differences profoundly impact the prevalence of sunny, rainy, snowy, and cloudy weather.

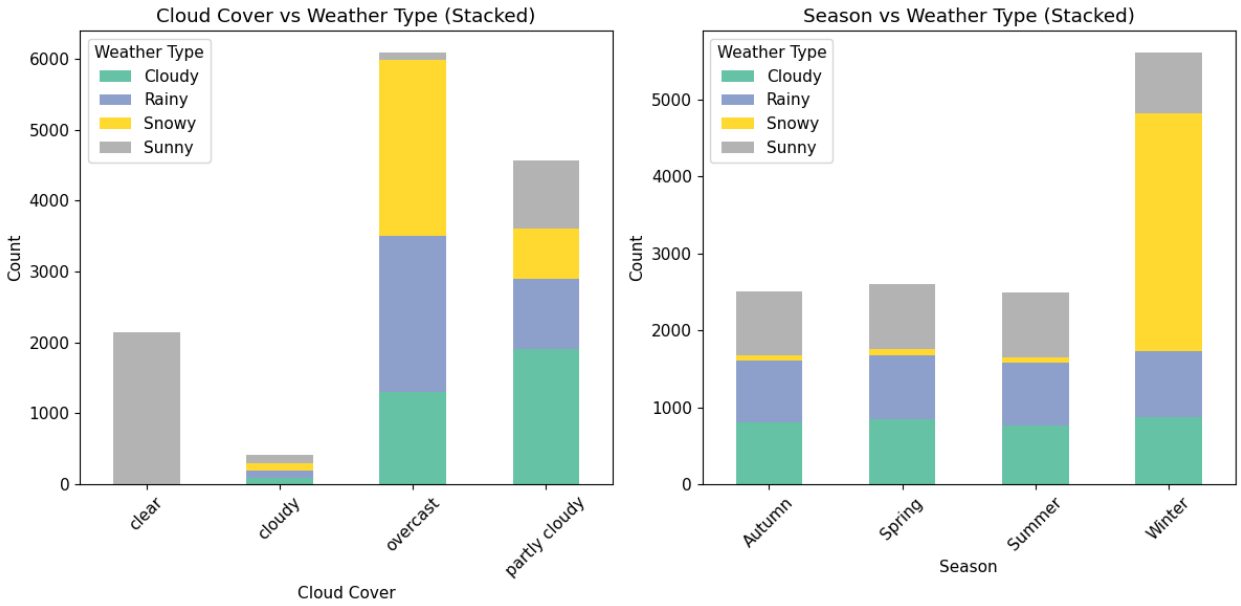

Figure 3: Categorical Features vs. Weather Type

- Continuous features vs. Weather Type. Figure 4 below shows some information about continuous features. Sunny weather typically has higher temperatures and UV index values, while snowy weather shows the lowest in both categories, indicating the intensity of sunlight and ambient temperature variations with weather conditions. Rainy weather is characterized by high humidity and precipitation levels, which decrease significantly in sunny conditions and are even lower in snowy weather, highlighting the moisture content associated with different weather types. Rainy and snowy weather is associated with higher wind speeds and reduced visibility compared to other weather types.

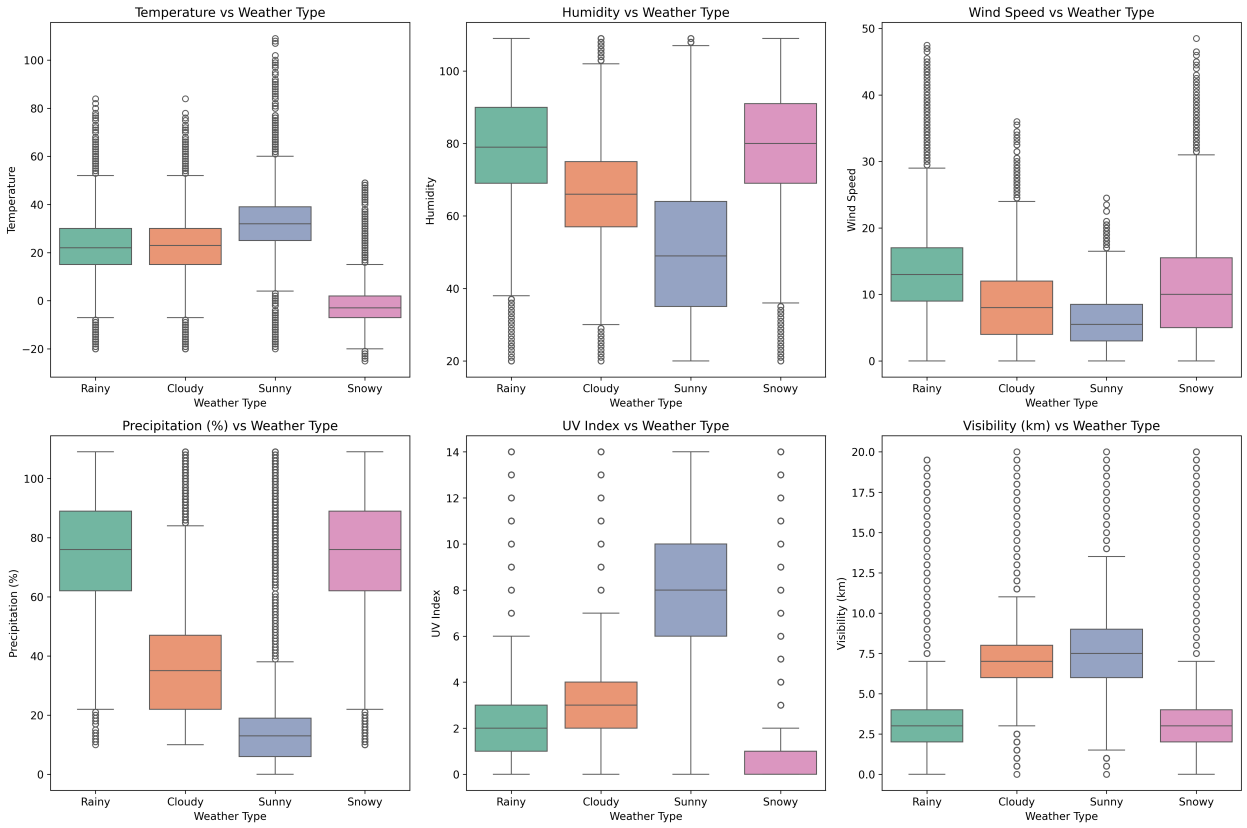

Figure 4: Continuous Features vs. Weather Type

3. Methods

3.1 Splitting

I used ‘GroupShuffleSplit’ and ‘GroupKFold’ to split the dataset. First, the dataset is split into other (80%) for training and validation and testing (20%) by group shuffle split to ensure the points with same ‘Location’ value are split into the same set to avoid data leakage. Additionally, Group K-Fold, a cross-validation strategy, is used during model training to assess performance consistently across multiple splits of the training data by using each split as validation set once. And in this way, to make sure that all points having same ‘Location’ value are in the same set.

3.2 Preprocessing

- For continuous features, I used ‘StandarScaler’ to to ensure they have a mean of 0 and a standard deviation of 1. This step is critical for algorithms sensitive to feature scaling.
- Since I used XGBoost, I translated target variables using label encoding.
- Since ‘Location’ is used as the key for GroupShuffleSplit and GroupKFold, there are no features needed to be one-hot encoded.
- I also checked missing values, but there does not contain any of them. After that, I combined a column transformer pipeline to preprocess dataset.

3.3 Machine learning pipeline

- Group shuffle split into other and test set
- Preprocessing
- Group K-Fold Cross-Validation in other set
- ParamsTuning (GridSearchCV)
- Testing
- Store scores and find the best model with best parameters

3.4 Metric for evaluating model’s performance

Since the class distribution in the dataset is relatively balanced, **accuracy** provides a straightforward measure of overall model performance. For imbalanced datasets, other metrics such as F1-Score, precision, and recall could be used. I also checked the F1-Score showing the similar situation as accuracy.

3.5 Algorithms

I tried **Logistic Regression**, **Support Vector Machines (SVM)**, **Random Forest**. **KNN** and **XGBoost,** and the parameters and the best combinations I tuned as shown as follows (Figure 5):

Algorithms	Parameters
Logistic Regression	C = [0.01, 0.1, 1, 10, 100] penalty = ['l2'] solver = ['lbfgs', 'sag', 'newton-cg']
Random Forest	max_depth = [5, 10, None] max_features = ['sqrt', 'log2', None] min_samples_split = [2, 5, 10]
SVM	C = [0.01, 0.1, 1, 10] gamma = [0.001, 0.01, 0.1, 1, 10, 100] kernel = ['rbf', 'sigmoid']
XGBoost	learning_rate = [0.01, 0.05, 0.1, 0.3] reg_lambda = [0e0, 1e-2, 1e-1, 1e0, 1e1, 1e2] max_depth = [5, 6, 10, None]
KNN	n_neighbors = [3, 5, 7, 10] weights = ['uniform', 'distance'] metric = ['euclidean', 'manhattan', 'minkowski']

Figure 5: Algorithms and Parameters

3.6 Considerations

- Splitting Strategy: Group shuffle split is crucial to prevent data leakage and multiple random states were used to ensure robustness.
- Preprocessing: Proper handling of categorical and numerical features ensures compatibility with all models.
- Algorithm Selection: A mix of linear and non-linear models was used to compare performance.

- Non-deterministic Algorithms: like randomized algorithms, for example, Random Forest and XGBoost, introduce stochasticity so that multiple runs are averaged to account for variability.
- Hyperparameter Tuning: Parameters were chosen based on their potential impact on model performance and computational cost.

4 Results

4.1 Scores

The dataset is nicely balanced by target feature, so I used Accuracy as my score. And the baseline score is 0.25. The comparison of models with baseline score is shown below in Figure 6.

Each score of models I trained is better than baseline. The best one is Random Forest (0.9146), which is slightly better than XGBoost (0.9125).

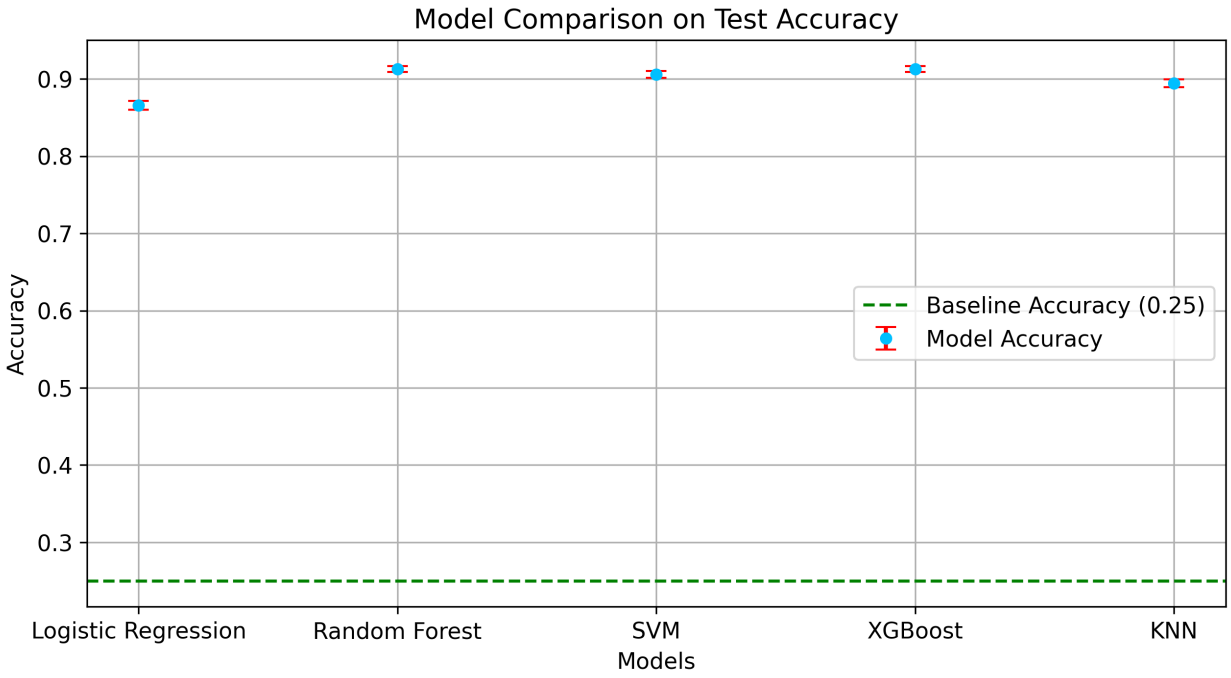

Figure 6: Model Comparison on Test Accuracy

The best combination and score of each model are shown in Figure 7.

Algorithms	Best Combinations	Accuracy
Logistic Regression	C = 10 Penalty = l2 Solver = lbfgs	0.8649 ± 0.0122
Random Forest	max_depth = None Max_features = sqrt Min_sample_split = 10	0.9146 ± 0.0065
SVM	C = 10 gamma = 0.1 Kernel = rbf	0.9090 ± 0.0073
XGBoost	learning_rate = 0.1 reg_lambda = 0.01 Max_depth = 6	0.9125 ± 0.0063
KNN	n_neighbors = 3 weights = uniform metric = manhattan	0.8990 ± 0.0086

Figure 7: Best Combination and Score of Each Model

The confusion matrix of Random Forest is shown in figure 8.

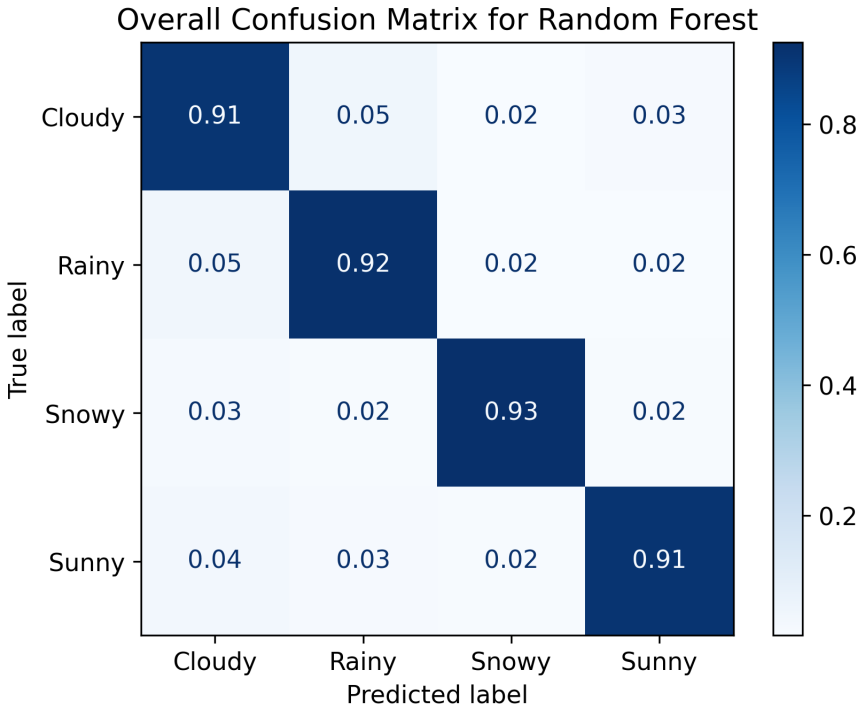

Figure 8: Confusion Matrix of Random Forest

4.2 Global Importance

Figure 9 to 11 show the importance ranking by different methods. Figure 9 is about feature_importance_ (accumulation of the impurity decrease within each tree[3]), figure 10 is perturbation importance, and figure 11 is SHAP global importance. These figures showed that temperature, precipitation and visibility are the 3 most important features while temperature is way more important than any other features. Humidity might be the least important feature.

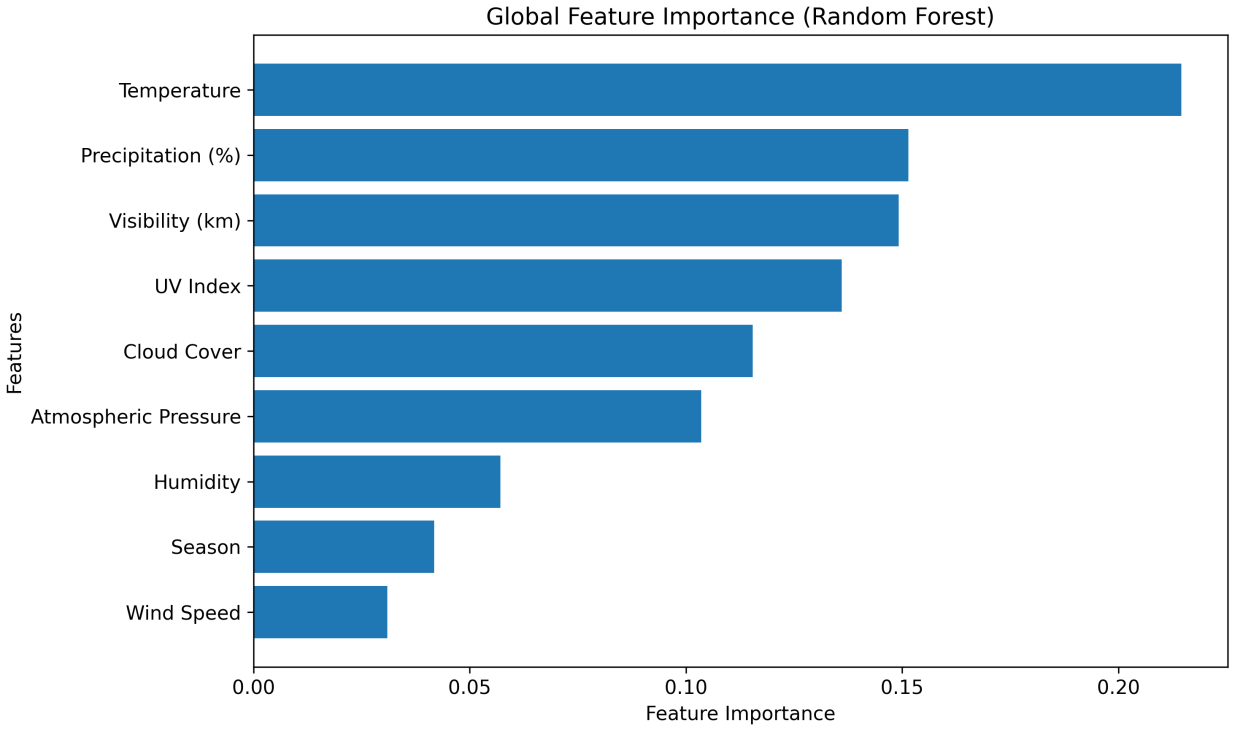

Figure 9: Global Importance by 'feature_importance_'

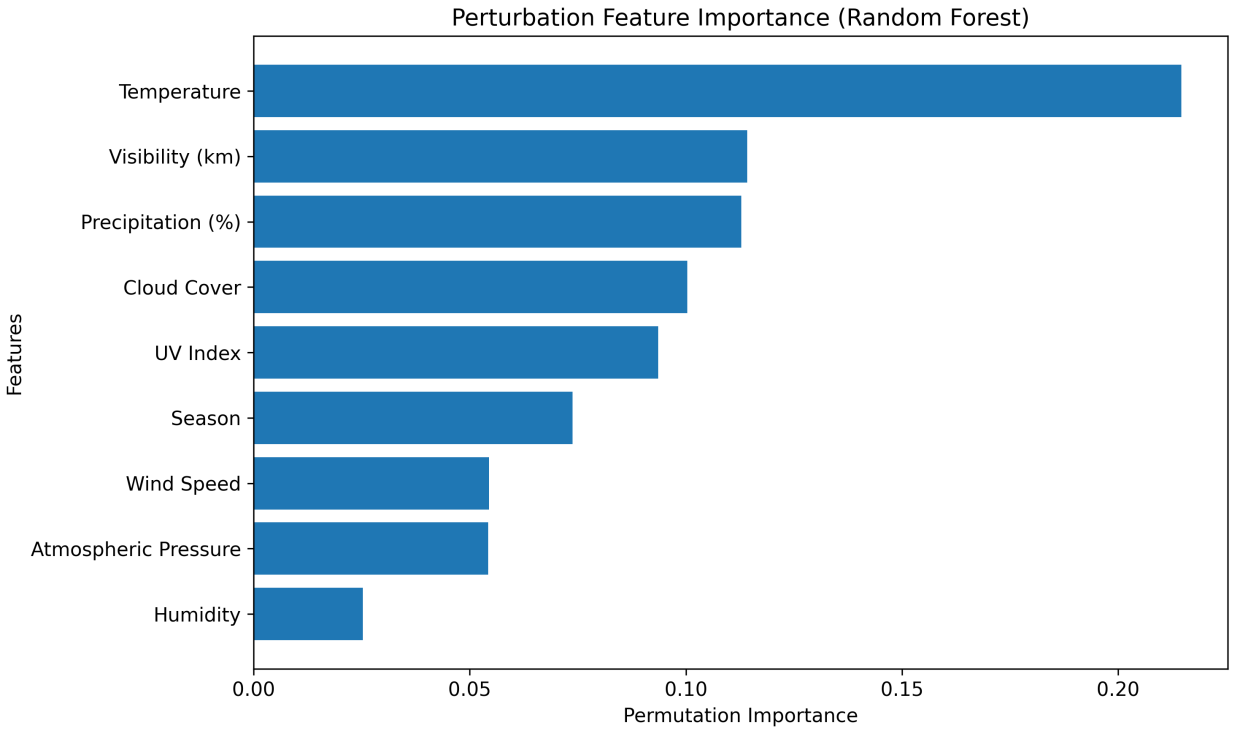

Figure 10: Global Importance by Perturbation

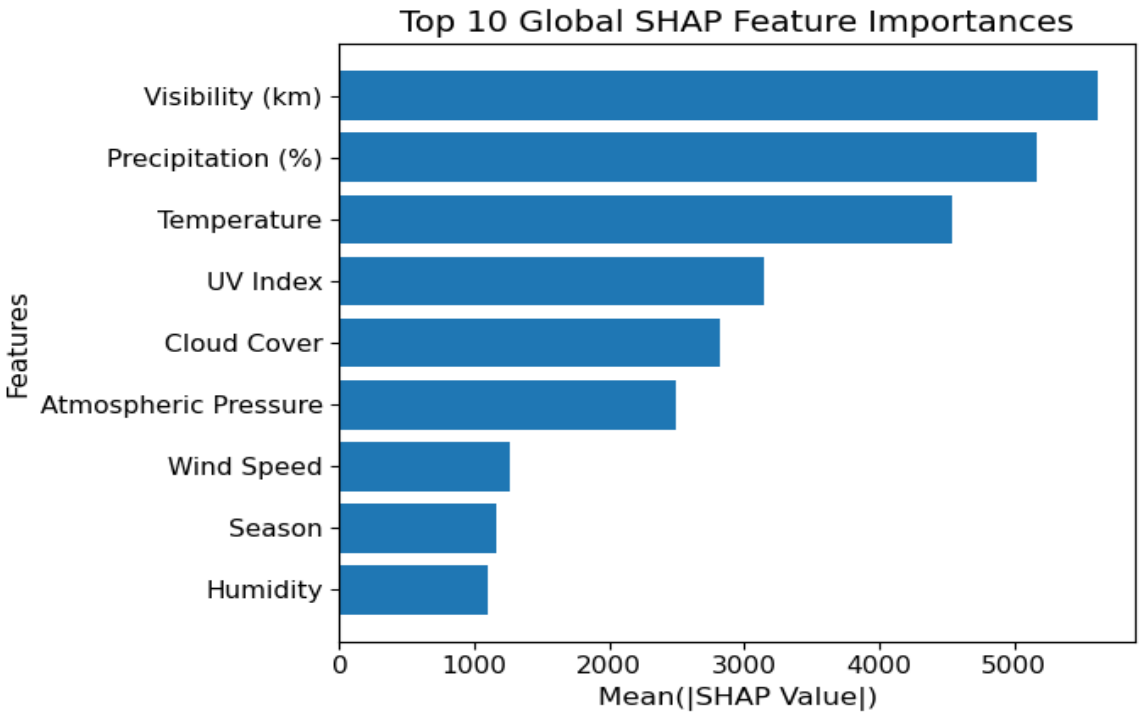

Figure 11: Global Importance by SHAP

4.2 Local Importance

First, I investigated single points. The two pictures in figure 12 shows the feature importance of two points. And the most important features are Cloud Cover, Precipitation which has negative impact, and Temperature has positive impact.

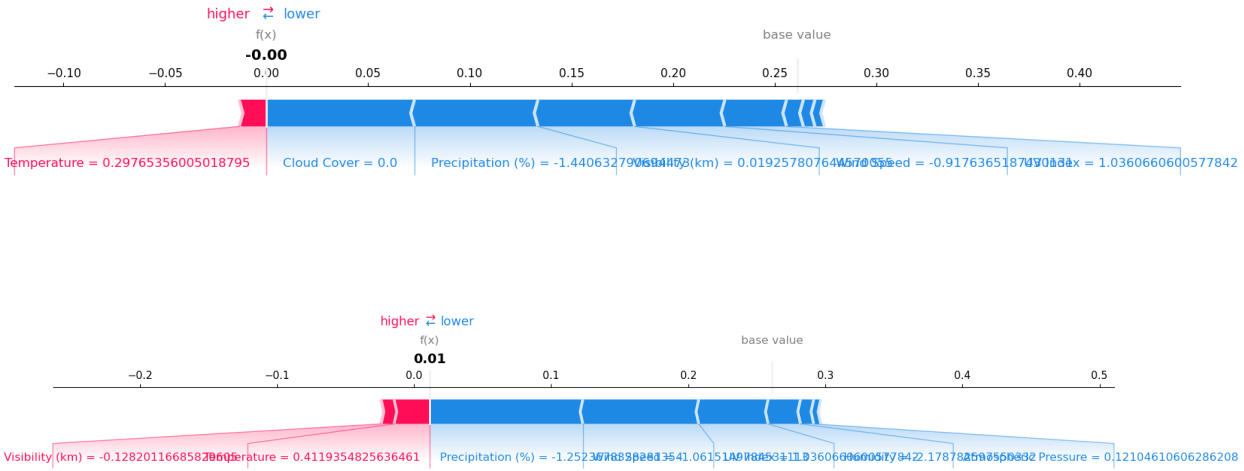

Figure 12: Local Importance by SHAP

Then I did Temperature value vs. SHAP value, and UV index dependence plot. Figure 13 shows that higher temperatures generally increase the prediction (positive SHAP values), while lower temperatures decrease it (negative SHAP values). The color represents the UV Index, which interacts with temperature. At higher temperatures, a higher UV Index further strengthens the positive impact on predictions, while at lower temperatures, the UV Index tends to be lower, aligning with a reduced prediction effect. This suggests that both temperature and UV Index jointly influence the model's output.

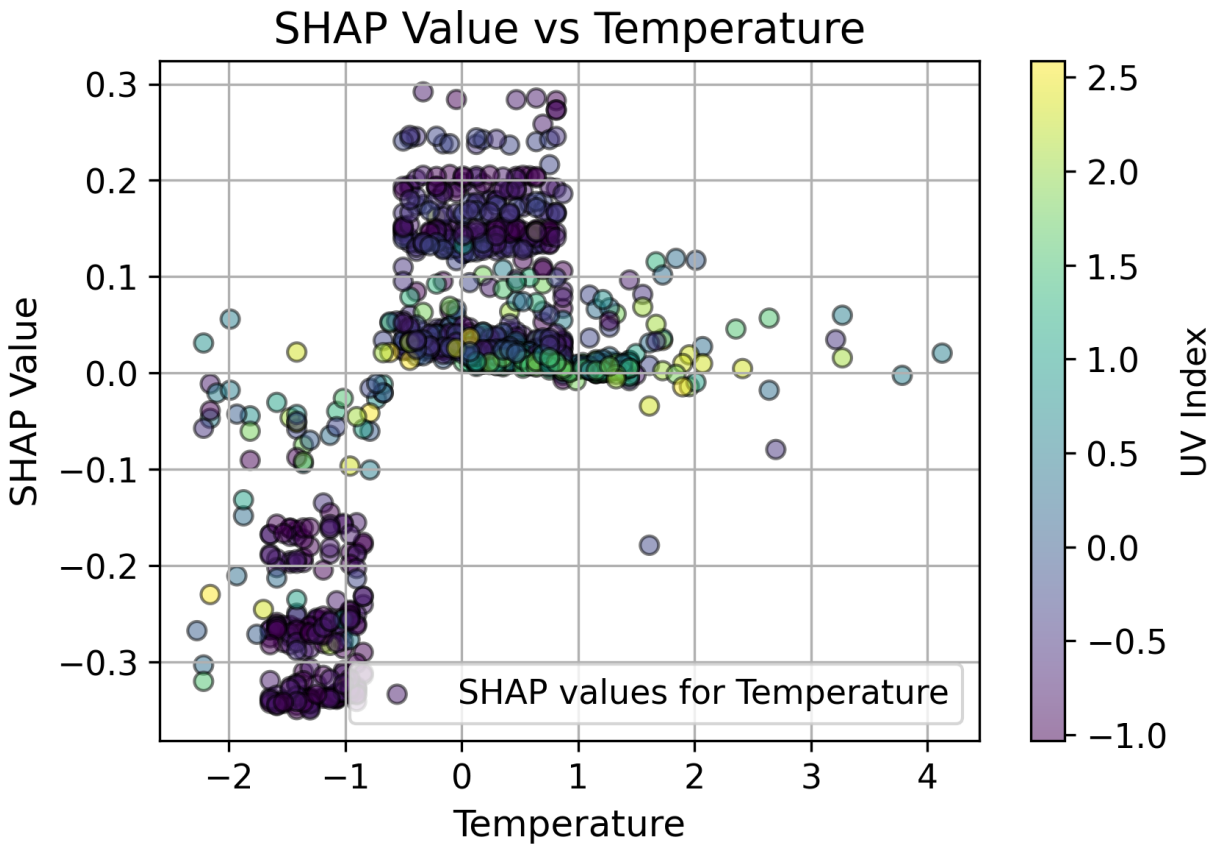

Figure 13: SHAP Values vs. Temperature, depending on UV Index

4.3 Findings

Although XGBoost is an upgraded version of Random Forest, its performance on this non-i.i.d. dataset is not as good as Random Forest, which might be related to the training parameters.

Additionally, it's surprising that **humidity** turned out to be the least important feature compared to others.

Furthermore, when using XGBoost for training, it's important to note that target variables need to be converted into numeric form, as XGBoost cannot process strings, unlike other methods.

The best model is Random Forest, with max_depth = None, Max_features = sqrt, Min_sample_split = 10, n_estimators = 100.

5 Outlook

To improve the model, I could:

- Refine hyperparameter tuning. Perform a more extensive hyperparameter search using advanced techniques like Bayesian Optimization or Random Search, instead of relying solely on Grid Search.
- Try deeper methods. Experiment with more advanced models such as Gradient Boosting Machines (LightGBM) or Neural Networks, which might better capture non-linear relationships in the data.
- More diverse data. Collect data from diverse sources or time periods to reduce biases and improve generalizability. Increase the dataset size to capture more variability, especially if the dataset is small.
- Get more features and values. Gather additional environmental factors that might have a significant impact on predictions. Also, expand the weather types, making it a better and stronger weather predictor.

Weak spots:

- Limited Feature Understanding. Some features, such as humidity, showed low importance, which might indicate poor feature quality or missing interactions.
- Potential Overfitting. Complex models like XGBoost and Random Forest can overfit if not tuned carefully, especially on smaller datasets or noisy features.

Reference

(1) https://www.kaggle.com/datasets/nikhil7280/weather-type-classification/data

(2) https://www.kaggle.com/code/ihabsherbiny/weather-classification-with-3-models#Decision-Tree-Classifier

(3) https://scikit-learn.org/1.5/auto_examples/ensemble/plot_forest_importances.html