Problem Statement

The weather has a significant impact on the agricultural industry and because of that, being able to predict it helps farmers in their day-to-day decisions such as how to plan efficiently, minimize costs and maximize yields.

A major agricultural company needs you to help them maximize growth efficiency, save resources and optimize their production.

To achieve these things, the company needs to have an accurate weather prediction algorithm that will improve their decision-making on typical farming activities such as planting and irrigating.

Using historical weather information from their region, ***can you predict what the weather will be like in the next few days****?*

Source

You now have a clear goal.

The goal 📖

Predict the next day's weather based on three labels.

- `N` — No rain
- `L` — Light rain
- `H` — Heavy rain

Let's now look at the data

The data 💾

```
🗁 train
  ├── region_A_train.csv
  ├── region_B_train.csv
  ├── region_C_train.csv
  ├── region_D_train.csv
  ├── region_E_train.csv
  ├── solution_format.csv
  └── solution_train.csv🗁 test
  ├── region_A_test.csv
  ├── region_B_test.csv
  ├── region_C_test.csv
  ├── region_D_test.csv
  └── region_E_test.csv
```

The data has been conveniently split into train and test datasets.

In each train and test, you're given weather data which consists of anonymized locations named region A through region E, which are all neighboring regions.

Here's a look at the first five rows of `region_A_train.csv`

The first thing you should notice is that the `date` column isn't a date but was anonymized to be some random value.

There is a total of 10 features, which are composed of temperature, precipitation, wind speed, wind speed direction, and atmospheric pressure

Then, looking at `solution_format.csv`

We can utilize the date column to join it with the training data and build a model.

Now that you have an idea about the goal and some information about the data given to you, it's time to get your hands dirty.

Get the data by registering for this data science competition and follow along with this article!

Code for this article → Google collab or Deepnote.

Installing libraries and models

First, install the weapon of our choice: `lightgbm`

Load Libraries

```
# essentials
import numpy as np
import pandas as pd

# plotting
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
matplotlib.rcParams['figure.dpi'] = 100
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.set(style="whitegrid")
%matplotlib inline

# ml
```

```
from sklearn.metrics import accuracy_score, recall_score,
ConfusionMatrixDisplay, classification_report, auc, precision_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import lightgbm as lgb
```

```
import joblib
```

Next, we load up some essential libraries for visualizations and machine learning.

Load the data

```
train_a = pd.read_csv("data/comp_datasets_train/region_A_train.csv")
train_b = pd.read_csv("data/comp_datasets_train/region_B_train.csv")
train_c = pd.read_csv("data/comp_datasets_train/region_C_train.csv")
train_d = pd.read_csv("data/comp_datasets_train/region_D_train.csv")
train_e = pd.read_csv("data/comp_datasets_train/region_E_train.csv")
```

```
test_a = pd.read_csv("data/comp_datasets_test/region_A_test.csv")
test_b = pd.read_csv("data/comp_datasets_test/region_B_test.csv")
test_c = pd.read_csv("data/comp_datasets_test/region_C_test.csv")
test_d = pd.read_csv("data/comp_datasets_test/region_D_test.csv")
test_e = pd.read_csv("data/comp_datasets_test/region_E_test.csv")
```

```
labels_df = pd.read_csv("data/comp_datasets_train/solution_train.csv")
```

Let's read in all the data we have.

Exploratory Data Analysis

```
train_a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 566 entries, 0 to 565
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   date                   566 non-null    object
 1   avg.temp               566 non-null    float64
 2   max.temp               566 non-null    float64
 3   min.temp               566 non-null    float64
 4   precipitation          566 non-null    float64
 5   avg.wind.speed         566 non-null    float64
 6   max.wind.speed         566 non-null    float64
 7   max.wind.speed.dir     566 non-null    object
 8   max.inst.wind.speed    566 non-null    float64
 9   max.inst.wind.speed.dir 566 non-null   object
 10  min.atmos.pressure     566 non-null    float64
dtypes: float64(8), object(3)
```

```
memory usage: 48.8+ KB
```

It’s time for the fun part, visualizing the data.

Join all the regions together

```
train_all = pd.concat([train_a ,train_b ,train_c ,train_d ,train_e],
keys=["A", "B", "C", "D", "E"])
train_alltrain_all = pd.concat([train_a ,train_b ,train_c ,train_d
,train_e], keys=["A", "B", "C", "D", "E"
```

Since all the region data share a primary key `date`, we can connect them with `concat()` in pandas and set the keys as the region names.

I don’t want the regions as the index, so we can reset the index and then rename some columns to get the data in the right shape.

Distribution of target class

```
train_all_lvls = train_all.reset_index()
train_all_lvls.rename(columns = {"level_0": "region"}, inplace=True)
train_all_lvls.drop(columns=['level_1'], inplace=True)
train_all_lvls.head()
```

Let’s first visualize our target class.

It appears we have in our hands an imbalanced class, as the `N` label is dominating the rest of the classes.

Why is this a problem?

The model will be biased towards classes with a larger amount of samples.

This happens because the classifier has more information on classes with more samples, so it learns how to predict those classes better while it remains weak on the smaller classes.

In our case, the label `N` will be predicted more than other classes.

Let's keep this in mind and move forward to more visualizations.

Plotting the features

We have ten features in each region.

It's time to cook up some plots.

Since all the regions are used to predict the next day's weather, let's see whether all the regions share similar patterns in the features and whether any outliers or anomalies exist.

From the plots, we see that the patterns in the data are very similar except for regions C, D, E
for `min.wind.speed` and `avg.wind.speed` which are on the lower scale.

Now that we've explored the data a little, we check for missing values in our data.

Missing values

Using my custom helper function, there seems to be a large amount of data missing for the `min.atmos.pressure` variable

Let's also use a heatmap to visualize the missing data for that column.

There are multiple ways we can deal with missing data.

Let's first look at the distribution of the column with missing values.

Since the distribution of the column is pretty normal, let's impute the missing data with the mean.

We then do the same for the test data (full code in the notebook)

Feature Preprocessing & Engineering

Converting data types

Let's first add the labels to our data.

Then we take a look at the categorical columns for our dataset.

We'll have to convert the categorical features, including the target variable to a numerical format.

Let's use scikit-learn's Label Encoder to do that.

Here's an example of using `LabelEncoder()` on the label column

Creating new weather features

Since we're given some weather features, there are interesting new features we can engineer.

One example is the Beaufort scale, which is "an empirical measure that relates wind speed to observed conditions at sea or on land."

Below is a function to do our feature engineering.

We can easily apply the feature engineering steps on the train and test dataset.

Prepare train data

Let's look at our train data so far.

It's time to pivot our data into the longer format, which means instead of the region being a column, each feature will have its own region, such as `avg.temp_A`, `avg.temp_B`, until `avg.temp_E` for the rest of the features.

This way the data will be in the right shape to build the model.

We can use the `pivot_table` function in pandas to achieve that.

The column names aren't ideal, so I wrote a function to fix that.

Here's what the train data looks like so far!

After we do the same to the test data, it's time to split the train data.

LightGBM likes it when the categorical features are given the categorical data type, so let's do that.

We can use `train test split` to split our data into the training and evaluation sets.

It's time to build the LightGBM model!

LightGBM

Let's create a base `lgb.LGBMClassifier` for a simple prediction

We can then `fit` the training data.

Then we call the `predict` function on the evaluation data

Model Performance

Let's see how a base LightGBM classifier did.

A 99% accuracy can be meaningless for an imbalanced dataset, so we need more suitable metrics like precision, recall, and a confusion matrix.

Confusion matrix

Let’s create a confusion matrix for our model predictions.

First, we need to get the class names and the labels that the label encoder gave so our plot can show the label names.

We then plot a non-normalized and normalized confusion matrix.

As you can see from the shade of the plot, our model is predicting the label `N` much more than others.

With the values you see in the plot above, we can compute some metrics that tell us how well our model is doing.

Classification Report

A classification report measures the quality of predictions from a classification algorithm.

It answers the question of how many predictions are True and how many are False.

More specifically, it uses True Positives, False Positives, True Negatives, and False Negatives to compute the metrics of precision, recall, and f1-score

Let’s break down the output.

`Precision` — What percent of your predictions were correct?
`Recall` — What percent of the positive cases did you catch?

`F1-score` — The weighted average of Precision and Recall
`support` —The number of occurrences of each given class

If you still have a hard time understanding precision and recall, read this great explanation.

Calculating the metrics

Taking class 0, which is the label `H` , let's calculate the precision, recall, and f1-score from the values in the confusion matrix

```
TP - True Positive | FP - False Positive
FN - False Negative | TN - True NegativePrecision = TP/(FP+TP) =
3/(3+4+2) = 3/9 = 0.33
Recall = TP/(TP+FN) = 3/(3+1+0) = 3/4 = 0.75F1-score = 2 *
(Recall * Precision) / (Recall + Precision)
         = 2 * 0.33 * 0.75 / (0.33 + 0.75)
         = 0.495 / 1.08
         = 0.458
```

Macro or weighted?

You might notice that there are three values for the overall F1-score `0.64`, `0.53`, `0.65` . Which value should you focus on?

In an imbalanced dataset where all classes are equally important, `macro average` is a good choice as it treats all classes equally.

If however, you want to assign greater weight to classes with more samples in the data, then the weighted average is preferred.

One more metric you can use is Precision-Recall, which is a useful measure of the success of prediction when the classes are very imbalanced.

Feature Importance

Let's also plot the feature importance to see which features matter more.

From the plot above, the wind speed features and precipitation are the key features that are good predictors.

The Beaufort scale feature we created seems to have very low importance, so it might be better to remove them.

Interestingly, `min.atmos.pressure` in region A is the most important, wheres `min.atmos.pressure` in other regions are among the lowest in importance.

Saving the model

When you're satisfied with your model, you can save it as a pickle file with `joblib`

Predict on test data

We have a simple model built. It's time to predict it on the test data.

Let's have a peek at the final submission file.

The labels are still encoded as numeric values; let's bring the actual label names back.

Since we already have the dictionary of the mapping of label names to numeric values, i.e. `'H' : 0`, we can reverse the dictionary above to map the numbers to the names

Save the prediction file.

Next steps

The base model won't be enough to make a good prediction; here are some next steps to improve upon the given approach.

1. More feature preprocessing and engineering
2. Use cross-validation to have a better measure of the performance.
3. Use Hyperopt or Optuna to tune the parameters of the LightGBM
4. Tune **class_weight** parameter of LightGBM directly to handle imbalance classes
5. Balance dataset by utilizing undersampling (taking a smaller sample from majority to match smaller sample) or resampling (using algorithms like SMOTE to augment dataset with fake data)
6. Test out other algorithms like KNN, SVM, XGBoost, Catboost, etc.

7. **Join the bitgrit discord server** to discuss the challenge with other data scientists