Forero Guaitero Kevin Andres

Lancheros Sanchez Christian Camilo

Delgado Jimenez Miguel Esteban

How Much Did It Rain? - Technical Report

Data Collection:

The data used in this study was sourced from the National Weather Service (NWS) and consists of radar and gauge measurements. This data was collected over a specific period and geographical region, providing a robust dataset for the analysis and prediction of rainfall. The following subsections describe the dataset, the collection methods, and the preprocessing steps taken to prepare the data for analysis.

The dataset comprises NEXRAD and MADIS data collected over 20 days between April and August 2014, across midwestern corn-growing states in the United States. This period was selected to capture the varying weather conditions typical of the corn-growing season. The dataset includes both training and test sets:

Train.zip: Contains radar observations at gauges in the Midwestern US over 20 days each month during the corn-growing season. Each observation is accompanied by a gauge measurement taken at the end of each hour.

Test.zip: Contains radar observations at gauges in the Midwestern US over the remaining 10/11 days each month of the same years as the training set. The task is to predict the gauge measurement at the end of each hour.

The radar data was collected using the NEXRAD network, which provides high-resolution radar observations, including reflectivity (`Ref`), specific differential phase (`Kdp`), and correlation coefficient (`RhoHV`). The gauge data, representing the actual rainfall measurements, was collected using rain gauges installed at various locations within the study area. The radar data includes both horizontal and vertical polarization measurements, enhancing the ability to differentiate between different types of precipitation and their characteristics.

Data Preprocessing:

We had a lot of problems with the dataset, since the dataset was full of noise, so we were forced to eliminate most of the records and polish many that would generate noise in the system, these were the steps we took:

The raw data was inspected for missing values and inconsistencies. Missing values were addressed by filling them with mean values.

Outliers were identified using statistical methods and visual tools such as violin plots and box plots. Values beyond three standard deviations from the mean were considered outliers and were removed to improve the dataset's representativeness.

The data was normalized to ensure that the features were on a similar scale, which is crucial for the performance of many machine learning algorithms.

We made these decisions based on the fact that about 80% of the dataset was incomplete data, so we first grouped them by hour and averaged them to reduce the number of records, we managed to reduce from approximately 14 million to 2 million data with greater consistency.

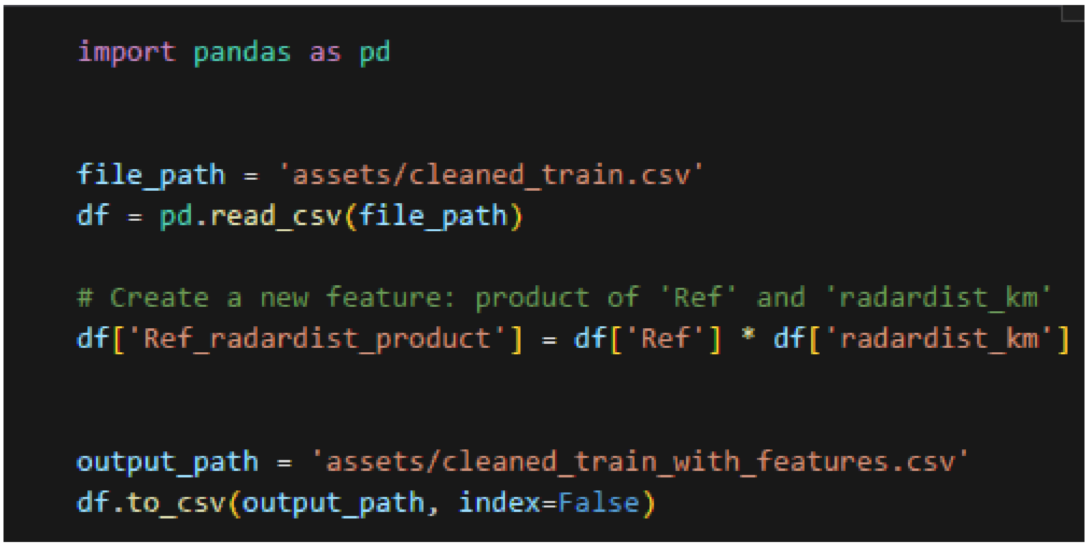
We then replaced the NaNs with 0 to minimize noise, because if we tried to delete records with a certain number of NaNs, we would have greatly affected the dataset, reducing our number of entries to less than 90% of the original data.

To justify the decisions, the three members discussed the decisions, but the most controversial one was to take that amount of data despite the noise that existed in the data, because we thought that despite having that massive amount of noisy data, we had a target that could work.

Finally, as a criterion to eliminate outliers, we chose to eliminate outliers with at least 3 STD distance, since there were outliers that exceeded 10 - 15 STD distance, which can be seen in the graphs placed in the paper.

Feature Engineering:

We have created a new feature called Ref radardist product, which is the product of the Ref (Radar Reflectivity) and radardist km (Radar Distance) columns. This feature can provide useful information by combining the intensity of radar reflectivity with the distance to the radar, which could help improve prediction models. Is given that the next code:



```
import pandas as pd


file_path = 'assets/cleaned_train.csv'
df = pd.read_csv(file_path)

# Create a new feature: product of 'Ref' and 'radardist_km'
df['Ref_radardist_product'] = df['Ref'] * df['radardist_km']


output_path = 'assets/cleaned_train_with_features.csv'
df.to_csv(output_path, index=False)
```

Model Deployment:

For model deployment, we considered several factors such as model performance, interpretability, and computational efficiency. Given the comparative performance of the Random Forest and Neural Network models in our experiments, we selected these two models for further exploration and potential deployment.

Random Forest Deployment

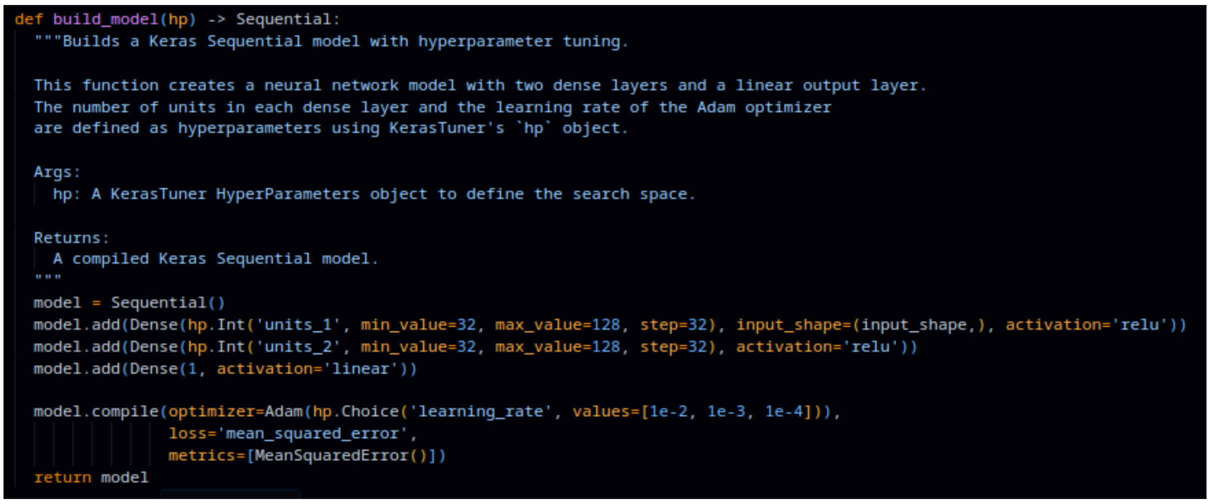
For the Random Forest model, we employed the RandomForestRegressor from the Scikit-learn library. To ensure comparability across features, data scaling was performed using StandardScaler from the same library.

Hyperparameter tuning was conducted manually, focusing on random_state and max_depth to explore their impact on model performance. To expedite the training process, the n_jobs=-1 parameter was utilized, allowing for parallel processing across all available CPU cores.

Neural Network Deployment

For the neural network we used tensorflow with keras. From the clean pandas data, this time we scaled it with StandardScaler, to improve the performance after a few runs with lousy results. Also, after several manual tests, we chose the network to be 3 layers, with more layers the results were worse. The first 2 layers of "relu" activation functions and the last one linear. As the results were still better for random trees, we wanted to implement an optimization by parameter variation, in this case Bayesian optimization to improve its metrics.

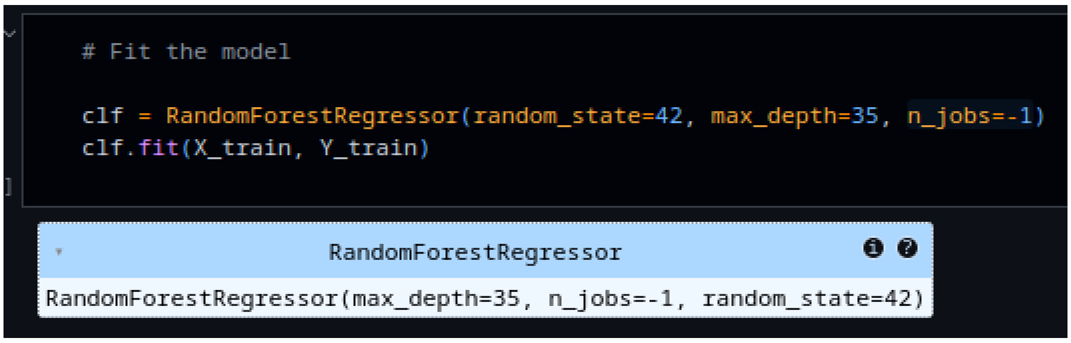
To implement hyperparameter optimization we create a function called "build_model" in order to iterate.



```python
def build_model(hp) -> Sequential:
  """Builds a Keras Sequential model with hyperparameter tuning.

  This function creates a neural network model with two dense layers and a linear output layer.
  The number of units in each dense layer and the learning rate of the Adam optimizer
  are defined as hyperparameters using KerasTuner's `hp` object.

  Args:
    hp: A KerasTuner HyperParameters object to define the search space.

  Returns:
    A compiled Keras Sequential model.
  """
  model = Sequential()
  model.add(Dense(hp.Int('units_1', min_value=32, max_value=128, step=32), input_shape=(input_shape,), activation='relu'))
  model.add(Dense(hp.Int('units_2', min_value=32, max_value=128, step=32), activation='relu'))
  model.add(Dense(1, activation='linear'))

  model.compile(optimizer=Adam(hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])),
                loss='mean_squared_error',
                metrics=[MeanSquaredError()])
  return model
```

Model Training:

Fit Random Forest:

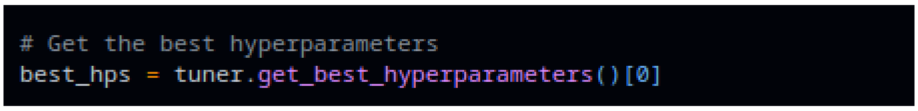


```python
# Fit the model

clf = RandomForestRegressor(random_state=42, max_depth=35, n_jobs=-1)
clf.fit(X_train, Y_train)
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=35, n_jobs=-1, random_state=42)
```

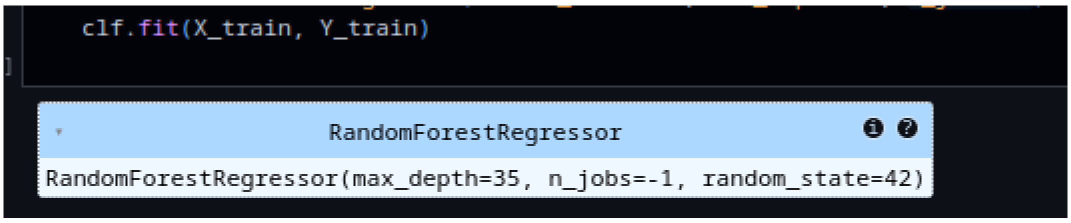
Fit ANN :

To train the network, the best hyperparameters are obtained first.



```python
# Get the best hyperparameters
best_hps = tuner.get_best_hyperparameters()[0]
```

To then train it with the best values found.



```python
clf.fit(X_train, Y_train)
```

```
RandomForestRegressor
RandomForestRegressor(max_depth=35, n_jobs=-1, random_state=42)
```

Model Evaluation:

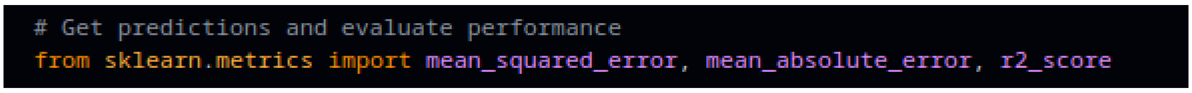
Metrics

1. Mean Squared Error (MSE): Measures the average squared difference between predicted and actual values.
2. Mean Absolute Error (MAE): Measures the average absolute difference between predicted and actual values.

3. R-squared: Represents the proportion of variance in the dependent variable explained by the model.

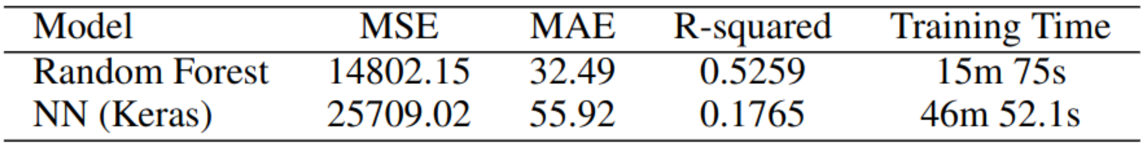
Model Selection:

We use the calculations of the metrics provided by sklearn to evaluate the models



```
# Get predictions and evaluate performance
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

The results were:



Model	MSE	MAE	R-squared	Training Time
Random Forest	14802.15	32.49	0.5259	15m 75s
NN (Keras)	25709.02	55.92	0.1765	46m 52.1s

Based on the provided metrics, the Random Forest model outperforms the Neural Network (NN) model in terms of Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared. The Random Forest model also exhibits significantly faster training time. The superior performance of the Random Forest model in this specific case can be attributed to several factors, including the nature of the data (data coming from a meteorological meter, with variable noise and interference), the complexity of the relationships between features (the complexity of the problem is chaotic) and the target variable, and the hyperparameter tuning process. It is important to note that the training time for the NN model might be improved by optimizing hyperparameters, utilizing more computational resources, or exploring different neural network architectures. Additionally, the full potential of the NN model might be realized with a larger dataset or different data preprocessing techniques.