COMPREHENSIVE ANALYSIS REPORT

TIME SERIES FORECASTING AND WEATHER CONDITION CLASSIFICATION

PREDICTA 1.0

By Warriors | P124
University of Ruhuna

Table of Contents

1. Summary

1.1 Overview of Approach and Key Findings

Methodology and Key Findings – Problem 1: Time Series Prediction

This competition aimed to predict the average temperatures for the first week of 2019 across 100 cities worldwide using historical weather data spanning from January 1, 2014, to December 31, 2018. We leveraged a Long Short-Term Memory (LSTM) neural network, known for its effectiveness in handling sequential data and capturing long-term dependencies. Our methodology encompassed preprocessing the data, engineering features, and training the LSTM neural network model for time series forecasting.

Key findings from the analysis

- The LSTM model was able to learn patterns from the historical weather data and provide reasonably accurate predictions.
- Data preprocessing steps, such as handling missing values and normalizing the data, were crucial in improving the model's performance.
- Feature engineering, including the introduction of date-based features, added valuable information for the model, enhancing the model's predictive power.
- By applying the model iteratively to each city's dataset, we ensured broad generalization across different geographical regions/locations.

Methodology and Key Findings – Problem 2: Classification Problem

This report details the process and findings of a project aimed at predicting weather conditions using machine learning. The primary task was to classify the **condition_text** for each day based on historical weather data. The methodology involved data preprocessing, feature engineering, model selection, hyperparameter tuning, and evaluating model performance.

Key findings from the analysis

- Handling missing data and effective preprocessing steps ensured a clean dataset, essential for accurate predictions.
- Engineered features like **temp_range** and **wind_product** significantly contributed to the model's performance.
- Extensive tuning of hyperparameters was crucial in optimizing model performance.
- The ensemble model leveraging Random Forest, Gradient Boosting, and XGBoost classifiers provided robust and accurate predictions, outperforming individual models.
- The final model demonstrated high accuracy in classifying weather conditions, indicating the effectiveness of the methodology.

1.2 GitHub Repository

The code used for this competition (both Problem 1 and Problem 2) is publicly available on GitHub repository: *Predicta 1.0 - Kaggle Competiton (P124)*

2. Problem 1: Time Series Prediction

2.1 Data Understanding and Preprocessing

Exploration of the Historical Weather dataset

The **historical_weather.csv** file provided comprehensive weather data for 100 cities over a five-year period. The dataset included following attributes.

- **city_id**: Unique identifier for each city.
- **date**: Date of the weather observation.
- **avg_temp_c**: Average temperature in Celsius.
- **min_temp_c**: Minimum temperature in Celsius.
- **max_temp_c**: Maximum temperature in Celsius.
- **precipitation_mm**: Precipitation in millimeters.
- **snow_depth_mm**: Snow depth in millimeters.
- **avg_wind_dir_deg**: Average wind direction in degrees.
- **avg_wind_speed_kmh**: Average wind speed in kilometers per hour.

Data Cleaning and Preprocessing steps

Handling Missing Values: Missing values in the dataset were handled using linear interpolation, which helps in maintaining the data integrity and continuity of the time series data. This method is particularly effective in time series data, maintaining the natural progression without introducing significant bias.

Date Conversion: The **date** column was converted to datetime format to facilitate the extraction of date-based features.

Feature Engineering: Additional features such as **year**, **month**, **day**, and **day_of_week** were created from the date column.

Normalization: The temperature data (**avg_temp_c**) was normalized using Min-Max scaling, transforming the **avg_temp_c** values to a range between 0 and 1. This step was essential for ensuring the stability and efficiency of the LSTM model during training.

2.2 Feature Selection and Engineering

Feature Engineering

Creation of Date-Based Features: The extraction of date-based features from the date column was a critical step in feature engineering. These features **- year**, **month**, **day**, and **day_of_week -** captured seasonal patterns and daily variations, which are vital for accurate weather prediction. By providing additional context, these features enabled the model to better understand and leverage the temporal structure of the data.

Normalization and Scaling: Normalizing the **avg_temp_c** values using Min-Max scaling ensured that the data was on a consistent scale, which is crucial for the LSTM model. This preprocessing step prevented any single feature from disproportionately affecting the training process, thereby improving model performance and stability.

Justification for Features

Temporal Features

- Year: Including the year helps capture long-term trends in temperature data, which may be influenced by factors like climate change or urbanization.
- Month: Monthly patterns are crucial in weather data to account for seasonal variations. For instance, temperatures in January are typically lower than those in July in many regions.
- Day: The day of the month can capture finer temporal granularity and potentially reveal intra-month trends or patterns.
- Day of Week: This feature can account for weekly cycles or patterns that may be present in the data, such as differences between weekdays and weekends in urban areas.

Normalized Temperature **(avg_temp_c)**: This is essential for the LSTM model to perform well, as neural networks typically train more efficiently and effectively when the input data is normalized. Using normalized average temperature directly captures the historical temperature patterns and temperature trends, which are fundamental to accurate time series forecasting.

2.3 Model Selection and Training

Model Description (LSTM - Long Short-Term Memory)

LSTM Layers: The model architecture includes two LSTM layers.. Each LSTM layer consists of 50 units (or neurons), which means it has the capacity to capture complex patterns in the data over time.

Return Sequences: The first LSTM layer has **return_sequences=True**, meaning it returns the full sequence of outputs for each input sequence, which is then fed into the second LSTM layer. This helps in capturing the temporal dependencies more effectively.

Dense Layer: Following the LSTM layers, a Dense layer with a single neuron is used. This layer is responsible for producing the final prediction, which is the average temperature for the next day in the sequence.

Activation Functions: The LSTM layers use the default activation functions suitable for LSTM cells (usually the sigmoid function for the gates and the tanh function for the cell state)

Hyperparameter Tuning

Sequence Length: The sequence length, set to 30 days, means the model uses the past 30 days of temperature data to predict the next day's temperature.

Epochs: The model is trained for 10 epochs. This number of epochs is selected to allow the model sufficient time to learn from the data while preventing overfitting.

Batch Size: A batch size of 32 is used. This size is a common choice that offers a good trade-off between computational efficiency and the ability to converge to a good solution.

Validation Split: During training, 20% of the training data is set aside for validation.

Early Stopping: Early stopping is employed to prevent overfitting. It monitors the validation loss and stops training if the loss does not improve for 5 consecutive epochs, indicating that the model has stopped learning general patterns and is starting to overfit to the training data. A patience of 5 epochs is chosen to provide the model with enough opportunity to improve even if it experiences temporary plateaus in validation loss.

2.4 Results and Discussion

Evaluation Metrics

Root Mean Squared Error (RMSE) used to measure the model's performance.

$$RMSE = \sqrt{\frac{1}{N}\sum_{I=1}^{N}(y_i - \hat{y}_i)^2}$$

Where;

- y_i is the actual average temperature
- $\hat{y}_i$ is the predicted average temperature
- N is the total number of predictions

Forecasting Results

Methodology: The LSTM model is used to predict the average temperatures for each city for the first week of January 2019 (January 1st to January 7th). This involves using the most recent 30-day sequence from the historical data to predict the temperature for January 1st, then updating this sequence with the predicted value to forecast January 2nd, and so on.

City-Specific Predictions: Each city has its own model trained on its historical weather data, ensuring that the model captures the unique weather patterns and trends of each city.

Post-Processing: The predicted temperature values are initially scaled between 0 and 1 due to normalization during preprocessing. After prediction, these values are scaled back to their original range using the inverse transform of the MinMaxScaler, providing meaningful temperature values in degrees Celsius.

Accuracy of Predictions: The LSTM model generally provided reasonable temperature predictions for the first week of 2019. The predictions align well with the observed temperature trends in many cases.

Variability in Performance: The model's accuracy varied across different cities. Cities with more stable and predictable weather patterns showed better prediction accuracy compared to those with highly volatile weather conditions.

Discussion

<u>Model Performance</u>

The model's performance varied significantly across different cities (city-specified variations), indicating the need for city-specific tuning and possibly different models for different cities. Some cities with more predictable weather patterns had lower prediction errors, while cities with more erratic weather showed higher errors.

<u>Insights</u>

The LSTM model effectively captured temporal dependencies in the historical temperature data. Its ability to remember long-term patterns and trends is crucial for time series forecasting tasks. LSTM networks are well-suited for sequential data, and their gating mechanisms help in retaining relevant information over long periods.

The model's performance is sensitive to the quality and quantity of the historical weather data. This is a limitation. Cities with incomplete or sparse data had higher prediction errors. Highly volatile weather conditions posed a challenge for the model. Capturing sudden changes in temperature requires more sophisticated approaches or additional contextual data (e.g., weather fronts, atmospheric pressure).

2.5 Conclusion

Summary of Findings

- The LSTM model effectively predicted average temperatures for short-term periods, particularly the first week of January 2019. The model's ability to handle sequential data and capture temporal dependencies was instrumental in generating accurate forecasts.
- The inclusion of date-based features (year, month, day, day of the week) provided the model with temporal context, helping it understand seasonal and cyclical patterns in the temperature data. Normalization of the temperature data ensured that the LSTM model could train efficiently and make accurate predictions, as it prevented issues arising from varying scales of input data.
- The model successfully leveraged temporal patterns in the historical data, indicating that weather data exhibits predictable trends that can be captured with appropriate time series models.
- The model's performance varied across different cities, highlighting the importance of city-specific tuning and the need for tailored models to account for unique weather patterns and data characteristics of each location.

Recommendations for Future Improvements

- Enhancing the dataset with additional weather variables such as humidity, wind speed, precipitation, atmospheric pressure, and other meteorological data can provide a richer context for the model.
- Usage of advanced architectures such as GRU (Gated Recurrent Units) and Hybrid Models (Combining LSTM with convolutional layers (CNN-LSTM))
- Adding dropout layers to the model can help prevent overfitting by randomly setting a fraction of input units to zero during training. Applying L2 regularization (weight decay) can help in penalizing large weights, promoting simpler models that are less likely to overfit.
- Usage of automated tools for Hyperparameter Tuning like GridSearchCV, RandomizedSearchCV and Bayesian Optimization
- Key hyperparameters that can be tuned include the number of LSTM units, learning rate, batch size, number of epochs, sequence length, and dropout rate. Optimizing these parameters can lead to improved model performance and generalization.

3. Problem 2: Classification Problem

3.1 Data Understanding and Preprocessing

Exploration of the Daily Weather Dataset

The dataset daily_data.csv contains various weather attributes recorded daily. Key columns include:

- **day_id**: Unique identifier for each day.
- **city_id**: Identifier for the city.
- **sunrise**: Time of sunrise.
- **sunset**: Time of sunset.
- **temp_c**: Temperature in Celsius.
- **humidity**: Humidity percentage.
- **wind_kph**: Wind speed in kilometers per hour.
- **gust_kph**: Gust speed in kilometers per hour.
- **condition_text**: Description of the weather condition.

Methods for Data Cleaning and Preprocessing

Handling Missing Values: The dataset was divided into training and test sets based on the presence of the **condition_text**. Rows with missing **condition_text** were separated to form the test set, while the rest constituted the training set.

Combining Data for Consistent Encoding: Both the training and test sets were combined temporarily to ensure consistent encoding of categorical features.

Categorical Encoding: The **city_id** column was encoded using LabelEncoder to convert categorical values to numerical values.

Time Conversion: The sunrise and sunset times were converted to total minutes from midnight for easier numerical handling.

Feature Engineering: Additional features such as **temp_range** (difference between **sunset** and **sunrise** times) and **wind_product** (product of **wind_kph** and **gust_kph**) were created to enrich the dataset.

Imputation of Missing Values: Missing values in numerical columns were imputed using the median strategy with SimpleImputer.

Normalization: Numerical features were normalized using StandardScaler to standardize the range of values.

Target Encoding: The target variable **condition_text** was encoded using LabelEncoder for classification purposes.

3.2 Feature Selection and Engineering

Features selected for Weather Condition Classification

- **city_id**
- **sunrise** (converted to minutes)
- **sunset** (converted to minutes)
- **temp_c**
- **humidity**
- **wind_kph**
- **gust_kph**
- **temp_range** (engineered feature)
- **wind_product** (engineered feature)

Explanation of Feature Engineering Decisions

Time Conversion (Sunrise and Sunset): The original **sunrise** and **sunset** times were in a format that included hours, minutes, and AM/PM designations. This format is not directly suitable for machine learning models that require numerical input. The times were converted to total minutes from midnight.

Temp Range (Temperature Range): The difference between **sunset** and **sunrise** (termed **temp_range**) captures the length of the daylight period, which can be an important factor in determining weather conditions. The feature was created by subtracting the **sunrise** time in minutes from the sunset time in minutes. A larger **temp_range** could indicate longer daylight, which might correlate with certain weather patterns like clearer skies or higher temperatures.

Wind Product (Wind Interaction Feature): Wind speed (**wind_kph**) and gust speed (**gust_kph**) are related but distinct measures of wind activity. Their interaction might provide additional insights into weather conditions, such as the presence of storms or calm weather. A new feature **wind_product** was created by multiplying **wind_kph** by **gust_kph**.

Categorical Encoding (City ID): The **city_id** column contains categorical data representing different cities. Machine learning models typically require numerical input, so categorical values need to be encoded. The LabelEncoder was used to convert **city_id** values into numerical labels. Each unique city was assigned a unique integer, allowing the model to learn patterns associated with different locations.

3.3 Model Selection and Training

Classification Algorithms used and approach to Model Parameter Tuning and Selection

Three primary classification algorithms were employed; **Random Forest Classifier (**An ensemble method using multiple decision trees), **Gradient Boosting Classifier** (An ensemble technique that builds trees sequentially, each trying to correct errors made by the previous trees) and **XGBoost Classifier** (An optimized implementation of gradient boosting, known for its performance and speed).

Hyperparameters for each model were tuned using GridSearchCV with cross-validation. The grid search explored various combinations of hyperparameters to identify the best settings for each classifier.

- Random Forest: Parameters such as n_estimators, max_features, max_depth, min_samples_split, and min_samples_leaf.
- Gradient Boosting: Parameters including n_estimators, learning_rate, max_depth, and subsample.
- XGBoost: Parameters like n_estimators, learning_rate, max_depth, subsample, and colsample_bytree.

Training Process

Data Preparation: The dataset was divided into training and testing sets based on the presence of the **condition_text**. Rows with missing **condition_text** formed the test set. Numerical features were scaled using StandardScaler to ensure they had a mean of zero and a standard deviation of one. This step is crucial for algorithms like Gradient Boosting and XGBoost that are sensitive to feature scaling.

Model Training: The best parameters identified by GridSearchCV were used to initialize the **RandomForestClassifier**. The model was then trained on the scaled training data (X_train and y_train). The **GradientBoostingClassifier** was initialized with the best parameters from the grid search and trained on the training data. The **XGBClassifier** was also trained using the optimal parameters determined from the tuning process.

Ensemble Model: Combining the strengths of different classifiers can improve overall model performance. An ensemble approach was chosen to leverage the diverse strengths of Random Forest, Gradient Boosting, and XGBoost. A **VotingClassifier** was used to create an ensemble of the three trained models. The voting strategy was set to 'hard', meaning that the final prediction was based on the majority vote from the individual classifiers.

Evaluation and Prediction: The ensemble model's accuracy was evaluated using the training set to ensure it was correctly capturing the patterns in the data. The trained ensemble model was used to predict the missing **condition_text** values in the test set. The predicted values were merged back into the original dataset. The final output, containing all **day_id** and the corresponding **condition_text**, was saved as submission.csv.

3.4 Results and Discussion

Performance Metrics

The accuracy of the models was evaluated based on the proportion of correct predictions

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

Analysis of classification results and model effectiveness

Ensemble Model Effectiveness

Voting Classifier ensemble model, combining Random Forest, Gradient Boosting, and XGBoost classifiers, achieved the highest accuracy among all tested models. The 'hard' voting strategy, which uses majority voting from the three classifiers, improved overall prediction robustness. The ensemble capitalized on the strengths of each individual model. Random Forest’s robustness, Gradient Boosting’s iterative refinement, and XGBoost’s computational efficiency contributed to a well-rounded predictive capability. Analyzing the misclassifications provided insights into areas where the ensemble model could be improved. Certain weather conditions with subtle differences (e.g., light rain vs. drizzle) were more prone to misclassification, suggesting the potential benefit of additional feature engineering or even more granular hyperparameter tuning.

Feature Importance

Feature importance scores from the Random Forest model highlighted the most influential features. **temp_range** and **wind_product** emerged as significant contributors, validating the feature engineering decisions. Gradient Boosting and XGBoost models provided similar insights into feature importance, reinforcing the critical role of engineered features. These models also indicated that **sunrise** and **sunset** times were highly predictive of certain weather conditions. Across all models, the engineered features and time conversions consistently ranked high in importance. This consistency underscored the effectiveness of the preprocessing and feature engineering steps.

Model Comparison - Strengths and Weaknesses

Each model had its strengths; for instance, Random Forest was particularly strong in handling high-dimensional data, while Gradient Boosting excelled in capturing complex patterns. XGBoost provided a balance between speed and accuracy. The ensemble model’s superior performance was a testament to the complementary nature of the individual classifiers. By combining their predictions, the ensemble mitigated the weaknesses of each model, leading to more accurate and reliable classifications.

Visualizations

Visual representations of confusion matrices for each model helped in understanding the specific areas of misclassification. These visual tools were instrumental in pinpointing which weather conditions were most frequently confused. Plotting the feature importance scores provided a clear visual of which features were driving the model predictions.

3.5 Conclusion

Conclusions

- The comprehensive approach using as effective methodology, combining thorough data preprocessing, strategic feature engineering, and robust model tuning, proved effective in solving the classification problem.
- Effective handling of missing data and preprocessing steps ensured a clean and comprehensive dataset.
- The success of the newly engineered features in enhancing model performance validated the importance of thoughtful feature engineering. These features significantly contributed to the model’s predictive accuracy.
- The ensemble model’s robustness and high accuracy highlighted the benefits of using multiple classifiers. This ensemble approach effectively captured the complexities of the dataset, leading to reliable predictions.
- The methodology demonstrated here is scalable and can be generalized to similar classification problems in other domains. The techniques and strategies used can be applied to different datasets and tasks, making this approach broadly applicable.

Recommendations for future enhancements

- Incorporate Additional Data: Including more weather attributes or external data sources could further improve model accuracy.
- Explore Advanced Models: Testing additional advanced models like neural networks or deep learning techniques might yield better results.
- Ensemble Enhancements: Implement more sophisticated ensemble techniques like stacking or blending
- Model Interpretability: Implementing methods for model interpretability, such as SHAP values, could provide deeper insights into the model's decision-making process.
- Data Augmentation and Enrichment: Integrate external data sources such as satellite imagery, geographic information, or real-time weather feeds.
- Operationalization and Deployment: Develop a robust model monitoring system to track the performance of the deployed models over time.