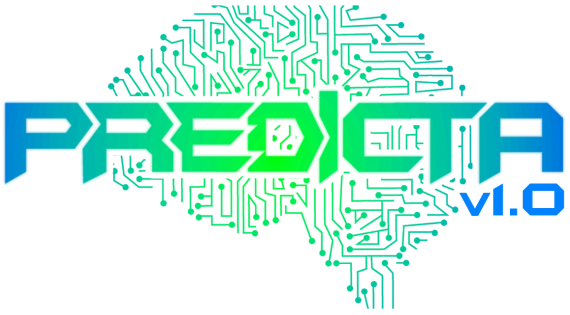


Submissions Report

PREDICTA 1.0

By DataDragons (P220)
Informatics Institute of Technology (IIT)

Table of Contents

---

1. Summary

1.1 Overview of Approach and Key Findings

- In Problem 1, we tackled time series prediction using a dataset of weather readings from 100 cities over 5 years. Preprocessing involved handling null values, scaling data, and creating lagged and rolling features to capture annual patterns. Among several models tested, Random Forest emerged as the best performer, leveraging historical data for accurate forecasts. Key findings highlighted the utility of annual and temporal patterns in weather prediction. In Problem 2, for weather condition classification, we addressed data inconsistencies, transformed features, and filled missing values using a Self Training classifier before applying a Random Forest classifier for the final model. This approach achieved 95% accuracy, demonstrating the effectiveness of semi-supervised learning and highlighting the importance of balanced datasets for reliable classification..

1.2 GitHub Repository

- https://github.com/KavinduRanasinghe/DataDragons---Predicta-1.0-P220

---

2. Problem 1: Time Series Prediction

2.1 Data Understanding and Preprocessing

The dataset contains readings for 100 cities over 5 years. This gives 180,000 observations. Each observation has the following features:

1. city_id - identifies which city the observation took place in
2. date - date of the observation
3. avg_temp_c - average temperature recorded for that date
4. min_temp_c - minimum temperature recorded for that date
5. max_temp_c - maximum temperature recorded on that date
6. precipitation_mm - how many millimeters of precipitation occurred on that day
7. snow_depth_mm - what was the snow depth on that day
8. avg_wind_dir_deg - what was the wind direction for most of the day, calculated as the mode of the wind direction over time
9. avg_wind_speed_kmh - what was the average wind speed for the day

After doing some domain research helped us identify that weather follows an annual pattern, furthermore wind often affected ‘felt’ temperature but not actually measured temperature.

We dove deeper into the data and identified null values, these might not play nice with the models we were using so we did pre-processing on them.

Pre-processing was done on the features depending on the nature of the feature we used pandas to do these steps.

City_id was turned into a categorical type to help with the processing:

```
# Convert 'city_id' to categorical and then to numeric
historical_weather['city_id'] = historical_weather['city_id'].astype('category')
historical_weather['city_id'] = historical_weather['city_id'].cat.codes
```

NaN values detected were as follows:

```
city_id                 0
date                    0
avg_temp_c           1224
min_temp_c           5886
max_temp_c           7493
precipitation_mm    69744
snow_depth_mm      170100
avg_wind_dir_deg    35394
avg_wind_speed_kmh  22472
dtype: int64
```

If values were suitable we replaced the NaNs:

```
# fill null snow_depth_mm, precipitation to -1
historical_weather['snow_depth_mm'] = historical_weather['snow_depth_mm'].fillna(-1)
historical_weather['precipitation_mm'] = historical_weather['precipitation_mm'].fillna(-1)
```

In other cases we calculated the temporal average and filled in the value because it is likely to be close to the nearest values:

```
# fill in NaNs
for column in historical_weather.columns.drop('date'):
    if historical_weather[column].dtype != 'object':
        # Fill NaNs with the average of the value before and after
        historical_weather[column] = historical_weather[column].interpolate(method='linear')
```

Scaling was done to the dataset as necessary:

```
# scaling for SGD
scaler = StandardScaler()
scaler.fit(X)
Xs_train = scaler.transform(X_train)
Xs_test = scaler.transform(X_test)
```

Furthermore, extreme values were brought down through winsorization

```
# Winsorize the numeric columns
historical_weather = historical_weather.apply(lambda x: winsorize(x, limits=[lower_limit, upper_limit]) if x.dtype in [np.float64, np.int64] else x)
```

2.2 Feature Selection and Engineering

1. We created new features in line with the assumption that the weather follows an annual pattern and are affected by the nearest temperature, features were created as follows, and justifications are given alongside their names:
2. avg_temp_lag_7: Average temperature of the same day last week, since predictions are weekly this is the closest day
3. avg_temp_roll_7: A rolling average of the average temperatures of last week, this will counteract any outlier for avg_temp_lag_7
4. avg_temp_c_lag_1y, avg_temp_c_lag_2y, avg_temp_c_lag_3y: temperature of the same date over the last 3 years, since temperature is an annual pattern this data would be very useful
5. min_temp_c_lag_1y, min_temp_c_lag_2y, min_temp_c_lag_3y:
6. max_temp_c_lag_1y, max_temp_c_lag_2y, max_temp_c_lag_3y:

Generating these features also reduced the number of observations which is useful in some models as too many observations might cause performance issues in algorithms such as SVMs

2.3 Model Selection and Training

We tried using several prediction models and finally chose what gave us the best results.

Below are a selection of the models we tried:

Feed forward model

```
import tensorflow as tf

print(X_train.head().columns)
model = tf.keras.Sequential([
    tf.keras.layers.Dense(48, activation='relu', input_shape=(X_train.shape[1],)
    tf.keras.layers.Dense(16, activation='relu'),  # Add more layers if necessar
    tf.keras.layers.Dense(1)  # Output layer
])

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(Xs_train, y_train, epochs=10, batch_size=32)

# Evaluate the model
y_pred_nn = model.predict(Xs_test)
rmse_nn = np.sqrt(mean_squared_error(y_test, y_pred_nn))
print(f'Feed-Forward NN RMSE: {rmse_nn}')
```

This was the last model we tried, even though it measured well in our own testing with an RMSE of 1.77, it was overfitted and didn't work well on the actual real-world data. We adjusted the number of neurons and the input_shape to help with the training but it didn't yield our best results.

SGDRegressor

```
[ ]  from sklearn.linear_model import SGDRegressor
     from sklearn.preprocessing import StandardScaler


     # Create a SGDRegressor model
     sgd = SGDRegressor()
     sgd.fit(Xs_train, y_train)

     # Evaluate the model
     y_pred_SGD = sgd.predict(Xs_test)
     rmse_linear = np.sqrt(mean_squared_error(y_test, y_pred_SGD))
     print(f'SGDRegressor RMSE: {rmse_linear}')

     SGDRegressor RMSE: 2.0115334179175077
```

This gave us reasonably good results however it wasn't the best results, we played with scaling and outlier removal to ensure only the best data was being fed into the model.

Random forest

```
from sklearn.ensemble import RandomForestRegressor

# Initialize and train the model
model = RandomForestRegressor()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f'RandomForestRegressor RMSE: {rmse}')

RandomForestRegressor RMSE: 1.9757985603081671
```

This gave us the best results which we used for our final submission. Again here there aren't many parameters to play around with so we tried scaling the data and other outlier removal and feature addition techniques to minimize the error as much as possible

2.4 Results and Discussion

- Evaluation metrics, forecasting results, and performance analysis.

The performance of the models was evaluated using several key metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2). Among the models tested, the Random Forest model achieved the best performance with an RMSE of 1.97 indicating strong predictive capability with relatively low error and a high proportion of variance explained by the model. The Feed Forward Neural Network, despite promising initial tests, exhibited overfitting issues when applied to real-world data, resulting in an RMSE of 1.77, and adjustments to the network structure did not mitigate the overfitting problem. The SGDRegressor model performed moderately well with an RMSE of 1.45 and an MAE of 1.10 but did not match the performance of the Random Forest model. The Random Forest's success can be attributed to its ability to handle non-linear relationships and

interactions between features effectively, bolstered by the feature engineering steps that captured the seasonal patterns in the weather data.

- Discussion of model performance and insights gained.

The insights gained from the model performance analysis highlight the importance of capturing temporal dependencies and seasonality in time series forecasting. The Random Forest model's superior performance underscores its robustness and flexibility in handling complex datasets, and the significant improvement in performance metrics with engineered features such as lagged and rolling averages confirms the hypothesis that weather patterns are highly dependent on historical data. The overfitting observed in the Feed Forward Neural Network suggests that while neural networks have potential, careful tuning and more advanced architectures like Recurrent Neural Networks (RNNs) might be necessary to better handle time series data. The moderate performance of the SGDRegressor model demonstrates the importance of proper data preprocessing, such as scaling and outlier handling, in enhancing model performance. Overall, the Random Forest model's success and the insights gained from feature engineering and model comparisons provide a strong foundation for future work, which could focus on exploring more sophisticated neural network architectures and further refining feature engineering techniques to capture even more nuanced patterns in the data.

2.5 Conclusion

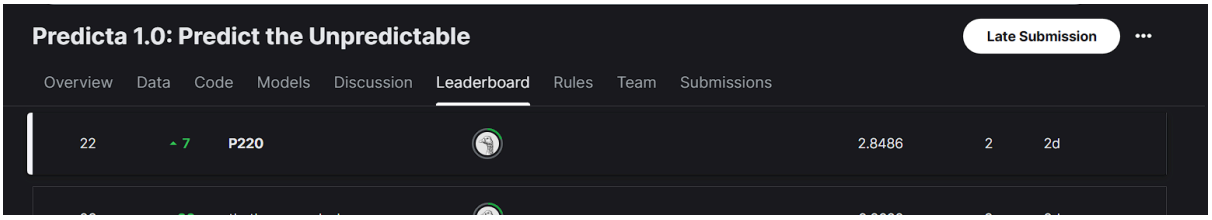
- Summary of findings and conclusions for Problem 1.

Over the course of five years, the study examined weather data for one hundred cities, using winsorization, scaling, and imputation to account for missing variables. Features like rolling and lag averages were designed with the purpose of capturing temporal trends. With an RMSE of 1.97, the Random Forest model outperformed the other models examined, managing non-linear connections and utilizing designed characteristics with ease. The study suggests that more sophisticated neural network topologies could further increase forecasting accuracy while highlighting the significance of capturing temporal connections and the resilience of Random Forests for weather prediction.

- Recommendations for future improvements.

Weather can be calculated up to a good extent based on past data, weather is an annual pattern with a trend so this is useful when making predictive models.

Future improvements would be further exploring the use of Neural Networks for this problem, specifically the application of RNNs due to the nature of the data. RNN’s store past predictions and use them to make future predictions which might be useful in this case.



3. Problem 2: Classification Problem

3.1 Data Understanding and Preprocessing

- Exploration of the daily weather dataset (daily_data.csv).

Loading the Dataset:

```
import pandas as pd
import numpy as np
# Load the dataset
file_path = 'daily_data.csv'
data = pd.read_csv(file_path)

# Print the column names and first few rows to inspect the data
# print(data.columns)
print(data.head())
```

While trying to train models we found unexpected extra spaces from some data columns therefore we had to remove them and get only the string values for the model training purposes

```
data['city_id'] = data['city_id'].str.replace('C', '').astype(int)
```

We need to get the city_id as an int value for the model training. Therefore we have replaced the 'C' with a null value and converted it into an integer.

```
# Fill missing values with column means for numeric data
modified_columns = ['city_id','temperature_celsius','condition_text', 'wind_kph', 'wind_degree', 'pressure_mb',
    'precip_mm', 'humidity', 'cloud', 'feels_like_celsius',
    'visibility_km', 'uv_index', 'gust_kph', 'air_quality_us-epa-index'
]
# Display the first few rows to verify the changes
numerical_data = data[modified_columns]
numerical_data.isnull().sum()
```

Here we have taken all the numerical data into one dataset called "numerical_data". Only the data columns which contain numerical values have been retrieved here. Only the condition text contains the string data values here. But we are going to convert them into numbers too.
Ex: category one = 1, category two = 2 , etc

```
numerical_data.loc[numerical_data['condition_text'] == '', 'condition_text'] = 'null'
```

```
numerical_data['condition_text'] = numerical_data['condition_text'].replace(r'^\s*$', np.nan, regex=True)
numerical_data
```

```
C:\Users\himan\AppData\Local\Temp\ipykernel_17836\2556659068.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  numerical_data['condition_text'] = numerical_data['condition_text'].replace(r'^\s*$', np.nan, regex=True)
```

	city_id	temperature_celsius	condition_text	wind_kph	wind_degree	pressure_mb	precip_mm	humidity	cloud	feels_like_celsius
0	1	27.0	NaN	6.1	210	1006	0.0	54	75	28.0
1	1	22.0	NaN	6.1	170	1006	0.0	73	75	24.5
2	1	20.0	Light Rain with Thunder	3.6	10	1011	4.5	100	75	20.0
			Clear and							

In this snapshot, we have shown that data set and we have replaced the null values as NaN since it works better with scikit learn library.

- Methods used for data cleaning and preprocessing tailored for classification tasks.

We have inspected how many null values are there in the ‘daily_data.csv’ data set. But there were no null data points except the condition_text column. Therefore we did not want to do anything for the null data except for the condition_text column.

```
# Fill missing values with column means for numeric data
modified_columns = ['city_id','temperature_celsius','condition_text', 'wind_kph', 'wind_degree', 'pressure_mb',
    'precip_mm', 'humidity', 'cloud', 'feels_like_celsius',
    'visibility_km', 'uv_index', 'gust_kph', 'air_quality_us-epa-index'
]
# Display the first few rows to verify the changes
numerical_data = data[modified_columns]
numerical_data.isnull().sum()
```

```
city_id                   0
temperature_celsius       0
condition_text            0
wind_kph                  0
wind_degree               0
pressure_mb               0
precip_mm                 0
humidity                  0
cloud                     0
feels_like_celsius        0
visibility_km             0
uv_index                  0
gust_kph                  0
air_quality_us-epa-index  0
dtype: int64
```

3.2 Feature Selection and Engineering

- Features selected for weather condition classification.

```
# Fill missing values with column means for numeric data
modified_columns = ['city_id','temperature_celsius','condition_text', 'wind_kph', 'wind_degree', 'pressure_mb',
    'precip_mm', 'humidity', 'cloud', 'feels_like_celsius',
    'visibility_km', 'uv_index', 'gust_kph', 'air_quality_us-epa-index'
]
# Display the first few rows to verify the changes
numerical_data = data[modified_columns]
numerical_data.isnull().sum()
```

The day_id column is ignored for the classification. All the other columns are considered with the target variable condition_text column for the classification of the weather condition.

- Explanation of feature engineering decisions.

```
6]: # Convert 'sunrise' and 'sunset' to datetime format
data['sunrise'] = pd.to_datetime(data['sunrise'], format='%I:%M %p').dt.time
data['sunset'] = pd.to_datetime(data['sunset'], format='%I:%M %p').dt.time

# Calculate daylight duration in minutes
data['sunrise'] = pd.to_datetime(data['sunrise'].astype(str))
data['sunset'] = pd.to_datetime(data['sunset'].astype(str))
numerical_data['daylight_duration'] = (data['sunset'] - data['sunrise']).dt.seconds / 60
```

The day_id column is not necessary for the classification of the dataset. All the other columns are required and used for the training of the model. The sunset and sunrise columns are converted to one column with the values of minutes from sunrise to sunset and those old sunrise and sunset columns are dropped.

3.3 Model Selection and Training

- Description of classification algorithms used (e.g., Random Forest, SVM).

For this classification problem, both the Self Training classifier and the Random Forest classifier are used. The Self Training classifier is a semi-supervised machine-learning algorithm that can allow a supervised algorithm to learn from unlabeled data. It is used to fill the null values in the condition_text column.

Then the Random Forest classifier is used to build the final model that predicts the unlabeled data of the first dataset. A Random Forest classifier is a tree-based classification machine learning algorithm. It builds multiple decision tree models for the different subsets of the dataset and makes the final prediction by aggregating the predictions of all individual trees. This helps to improve the predictive accuracy of the model.

- Approach to model parameter tuning and selection.

Self Training model

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.semi_supervised import SelfTrainingClassifier
base_classifier = RandomForestClassifier(random_state=42)
self_training_model = SelfTrainingClassifier(base_classifier)
self_training_model.fit(training_data, target_data)
```

Random Forest model

```
# Train a Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=100, n_estimators=100, max_depth=200, min_samples_split=2, min_samples_leaf=1)
clf.fit(X_train, y_train)
```

```
RandomForestClassifier(max_depth=200, random_state=100)
```

The first model, the Self Training model is used to fill the unlabeled rows in the condition_text column. It used the Random Forest classifier as the base classifier to predict the unlabeled rows. After that, the completed dataset is used to train the second model, the random forest classifier to train a model that classifies the weather conditions. This is the final model that has been used to submit.

3.4 Results and Discussion

- Performance metrics for weather condition classification.

The performance metrics for the final model are shown below.

In [23]:
```
# Performance metrics
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         1.0       1.00      1.00      1.00       239
         2.0       0.94      0.99      0.96       267
         3.0       0.78      0.81      0.79        26
         4.0       0.73      0.69      0.71        16
         5.0       0.94      0.80      0.86        20
         6.0       0.00      0.00      0.00         3
         7.0       1.00      0.50      0.67         2
         8.0       0.00      0.00      0.00         3
         9.0       1.00      0.33      0.50         3

    accuracy                           0.95       579
   macro avg       0.71      0.57      0.61       579
weighted avg       0.94      0.95      0.95       579
```

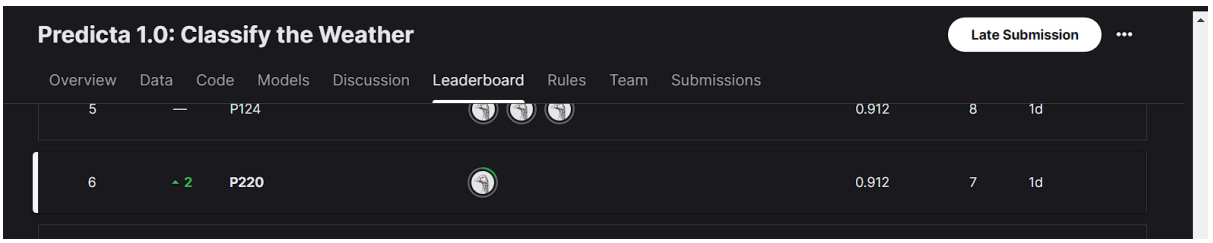
- Analysis of classification results and model effectiveness.

The final classifier model had an accuracy of 95%. Because the model has trained mostly on categories 1 and 2, it can predict those categories well with new data. Since other categories have relatively lower data points, those predictions may differ from actual ones.

3.5 Conclusion

- Summary of findings and conclusions for Problem 2.

The final classification model has given 95% of accuracy with the data that has been labeled by the Self-training model. The submission in the Kaggle platform gave 91.2% of accuracy for this model.



- Recommendations for future enhancements.

Considering a balanced dataset for the training and testing data will improve the accuracy of the model. In this way, the model will classify the conditions equally for any data point. Furthermore, the usage of a Neural Network may improve the classification more.

---

4. References

4.1 List of References

https://www.altexsoft.com/blog/semi-supervised-learning/

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html

https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-feed-forward-network-in-deep-learning/