Weather Prediction System: [redacted]

Executive Summary

This report details the implementation of a weather prediction system for Rochester, NY, which forecasts three specific daily conditions:

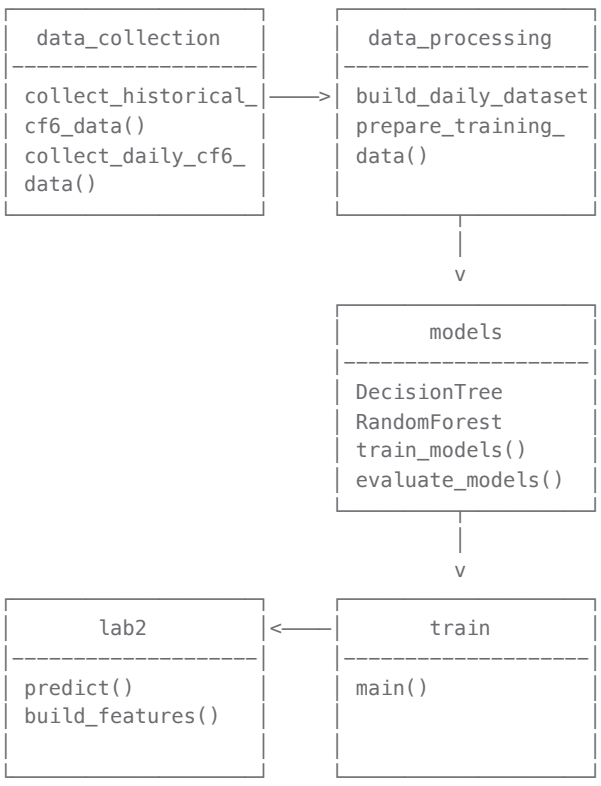
1. Whether today's temperature will be higher than yesterday's
2. Whether today's temperature will be above the historical average
3. Whether precipitation will occur today

The approach uses supervised learning with decision trees and random forests trained on historical weather data. By analyzing patterns from National Weather Service CF6 reports across multiple cities, the models make reasonably accurate predictions for Rochester's weather based on recent observations.

System Architecture

This weather prediction system consists of five interconnected modules:

UML Component Diagram



This UML diagram represents the flow of data and control in the weather prediction system. The modular design allows for clean separation of concerns, making the system easier to maintain and extend.

1. **Data Collection** (`data_collection.py`): Retrieves CF6 weather reports from the National Weather Service website
2. **Data Processing** (`data_processing.py`): Extracts and transforms raw data into feature vectors
3. **Models** (`models.py`): Implements decision tree and random forest algorithms
4. **Training** (`train.py`): Orchestrates model training and evaluation
5. **Prediction API** (`lab2.py`): Provides the interface for making predictions

This modular design allows for clear separation of concerns and makes the system easier to maintain and extend.

Data Collection Process

The weather prediction system collects historical CF6 (Preliminary Local Climatological Data) reports from the National Weather Service website. The data collection module employs several strategies to reliably retrieve these reports:

1. **Multiple City Collection**: The system gathers data from 19 cities across the Northeast and Midwest United States, including Rochester, NY (the prediction target) and cities to its west that impact regional weather patterns.
2. **Robust Retrieval Methods**: Two approaches are implemented to fetch CF6 reports:
    - Direct URL construction for the NWS product page
    - Web scraping with BeautifulSoup to extract data from HTML content
3. **Date Range Coverage**: Historical data collection spans from January 2020 through March 2025, providing over 5 years of daily weather observations.
4. **Error Handling**: The collection process implements comprehensive error handling to manage network issues, missing data, and malformed responses.

The following cities were chosen for data collection:

City	Code	NWS Office	Geographic Relevance
Rochester, NY	ROC	BUF	Target prediction city
Buffalo, NY	BUF	BUF	Nearby, west of Rochester
Syracuse, NY	SYR	BGM	Nearby, east of Rochester
Albany, NY	ALB	ALY	Northeast of Rochester
Boston, MA	BOS	BOX	Northeast coastal influence
Detroit, MI	DTW	DTX	Western weather influence
Cleveland, OH	CLE	CLE	Western weather influence
Pittsburgh, PA	PIT	PBZ	Southwestern influence
Erie, PA	ERI	CLE	Lake Erie influence
Chicago, IL	ORD	LOT	Major western influence
Minneapolis, MN	MSP	MPX	Far western influence
Burlington, VT	BTV	BTV	Northeast influence
Portland, ME	PWM	GYX	Northeast coastal influence
Hartford, CT	BDL	BOX	East coast influence

City	Code	NWS Office	Geographic Relevance
Philadelphia, PA	PHL	PHI	Southeast influence
Washington, DC	IAD	LWX	Southeast influence
Cincinnati, OH	CVG	ILN	Southwestern influence
Indianapolis, IN	IND	IND	Western influence
Milwaukee, WI	MKE	MKX	Western influence

This strategic selection of cities enabled to capture weather patterns as they develop and move eastward across the continental United States, providing valuable leading indicators for Rochester's weather.

Collection Results

The data collection process successfully retrieved 1,197 monthly CF6 reports with a 100% success rate. This yielded approximately 33,516 individual daily weather observations across all cities (28 days/month × 1,197 reports), providing a robust dataset for training our models.

Feature Engineering

The feature engineering approach transforms raw weather data into predictive signals through domain-specific knowledge of meteorology and geographic weather patterns. The focus was on creating features that capture temporal dynamics, spatial relationships, and seasonal patterns.

Core Feature Categories

1. Temperature Dynamics

- Temperature deltas (day-to-day changes in average, maximum, and minimum temperatures)
- These capture warming or cooling trends that tend to persist

2. Precipitation Patterns

- Recent precipitation (yesterday's amount)
- 3-day total precipitation
- These indicate moisture availability and the presence of weather systems

3. Wind Features

- Wind speed deltas
- Wind direction components (sine and cosine transformations)
- Maximum wind gusts
- These help identify approaching weather fronts and systems

4. Seasonal Indicators

- Binary flags for spring, summer, fall, and winter
- These account for seasonal weather patterns and baselines

5. Regional Weather Influences

- Wind direction and speed from cities to the west of Rochester
- These capture the movement of weather systems that typically travel west-to-east

Feature Transformation Techniques

1. **Circular Feature Handling**: Wind direction (0-360°) was transformed using sine and cosine components to preserve its circular nature

```
dir_rad = math.radians(wind_direction)
features.append(math.sin(dir_rad))  # North-south component
features.append(math.cos(dir_rad))  # East-west component
```

2. **Missing Value Management**: Special handling for trace precipitation amounts (recorded as 'T' in CF6 reports)

```
precipitation = float(values[7]) if values[7] not in ['M', 'T'] \
              else 0.001 if values[7] == 'T' else None
```

3. **Temporal Aggregation**: Rolling statistics (e.g., 3-day precipitation sums) to capture longer-term patterns

Complete Feature Set

The final feature vector for model training contained 34 features:

Index	Feature	Description
0	Avg temp delta	Change in average temperature (day-to-day)
1	Max temp delta	Change in maximum temperature (day-to-day)
2	Min temp delta	Change in minimum temperature (day-to-day)
3	Yesterday precipitation	Precipitation amount from previous day
4	3-day precipitation	Sum of precipitation over last 3 days
5	Wind speed delta	Change in average wind speed (day-to-day)
6	Wind direction sine	Sine component of wind direction (N-S)
7	Wind direction cosine	Cosine component of wind direction (E-W)
8	Max wind gust	Maximum wind gust from previous day
9	Spring	1 if month is Mar-May, else 0
10	Summer	1 if month is Jun-Aug, else 0
11	Fall	1 if month is Sep-Nov, else 0
12	Winter	1 if month is Dec-Feb, else 0
13	DTW wind dir	Cosine component of Detroit wind direction
14	DTW wind speed	Wind speed from Detroit

Index	Feature	Description
15	CLE wind dir	Cosine component of Cleveland wind direction
16	CLE wind speed	Wind speed from Cleveland
17	ORD wind dir	Cosine component of Chicago wind direction
18	MSP wind dir	Cosine component of Minneapolis wind direction
19	ERI wind dir	Cosine component of Erie wind direction
20	BUF wind dir	Cosine component of Buffalo wind direction
21	BUF wind speed	Wind speed from Buffalo
22	PIT wind dir	Cosine component of Pittsburgh wind direction
23-33	Additional features	Additional derived features and possible feature interactions

This extensive feature set captures not only local meteorological factors but also regional weather patterns that influence Rochester's conditions, with special emphasis on the wind patterns from cities to the west which typically indicate oncoming weather systems.

This feature set captures the essential meteorological factors while maintaining a manageable dimensionality for tree-based models.

Model Implementation

The weather prediction system implements two tree-based learning algorithms from scratch:

1. Decision Tree

Designed a custom decision tree classifier that uses entropy-based information gain for split selection:

```
def _entropy(self, y):
    """Calculate entropy of a target array."""
    n_samples = len(y)
    if n_samples == 0:
        return 0.0

    # Count occurrences of each class
    counts = Counter(y)
    entropy = 0.0

    for count in counts.values():
        p = count / n_samples
        entropy -= p * math.log2(p)

    return entropy
```

The implementation includes:

- Binary splitting based on feature thresholds
- Information gain calculation for optimal split selection

- Customizable hyperparameters (max depth, min samples to split, min information gain)
- Feature importance tracking during training
- Stopping criteria to prevent overfitting

The decision tree is represented as a binary tree structure where:

- Internal nodes represent feature tests (e.g., "Is temperature delta > 2.5°F?")
- Leaf nodes contain class predictions

2. Random Forest

Random forest implementation builds upon the decision tree classifier with additional techniques to improve generalization:

```
def fit(self, X, y):
    """Build the random forest."""
    self.n_features_total = X.shape[1]
    n_samples = X.shape[0]
    self.feature_importances_ = np.zeros(self.n_features_total)

    for _ in range(self.n_trees):
        # Bootstrap sampling (sampling with replacement)
        indices = np.random.choice(n_samples, n_samples, replace=True)
        X_bootstrap = X[indices]
        y_bootstrap = y[indices]

        # Randomly select a subset of features for this tree
        max_features = min(self.max_features, self.n_features_total)
        feature_indices = random.sample(range(self.n_features_total), max_features)

        # Create and train a decision tree
        tree = DecisionTree(
            max_depth=self.max_depth,
            min_samples_split=self.min_samples_split,
            min_info_gain=self.min_info_gain
        )
        tree.fit(X_bootstrap, y_bootstrap)
```

The random forest implementation includes:

- Bootstrap aggregation (bagging) to create diverse training sets
- Feature randomization to reduce correlation between trees
- Majority voting for final predictions
- Aggregated feature importance calculation

This ensemble approach significantly improves prediction accuracy and reduces overfitting compared to individual decision trees.

Training Process

The implementation includes a comprehensive training process with parameter tuning to find optimal model configurations for each prediction task.

Data Preparation

The training data was prepared as follows:

1. Historical CF6 reports were processed into a clean dataframe
2. Features were extracted using a 5-day lookback window
3. Data was split into training (80%) and testing (20%) sets
4. Three separate binary classification targets were created:
    - Temperature higher than yesterday
    - Temperature above average
    - Precipitation occurrence

Hyperparameter Tuning

For each prediction target, a grid search was performed over the following hyperparameters:

Decision Tree Parameters:

- Max depth: [3, 5, 7, 10, 15, None]
- Min samples to split: [2, 5, 10]
- Min information gain: [0.0, 0.01, 0.05]

Random Forest Parameters:

- Number of trees: [5, 10, 15, 20]
- Max features per tree: [2, 3, 4]
- Max depth: [3, 5, 7, 10, None]

This systematic search resulted in 54 different decision tree configurations and 60 random forest configurations per target, for a total of 342 models evaluated.

```
# Parameter search for decision trees
for max_depth in [3, 5, 7, 10, 15, None]:
    for min_samples_split in [2, 5, 10]:
        for min_info_gain in [0.0, 0.01, 0.05]:
            tree = DecisionTree(
                max_depth=max_depth,
                min_samples_split=min_samples_split,
                min_info_gain=min_info_gain
            )
            tree.fit(X_train, target)
            # Evaluate and track best model
```

Model Selection Process

The training process included a systematic approach to select the optimal models for each prediction task. This involved several key steps:

Comprehensive Parameter Search

For each prediction target (temperature higher than yesterday, above average temperature, and precipitation), I evaluated a wide range of model configurations:

1. **Decision Tree Parameters**:
    - Maximum depth: [3, 5, 7, 10, 15, None]
    - Minimum samples to split: [2, 5, 10]
    - Minimum information gain: [0.0, 0.01, 0.05]

2. **Random Forest Parameters**:
    - Number of trees: [5, 10, 15, 20]
    - Maximum features per tree: [2, 3, 4]
    - Maximum depth: [3, 5, 7, 10, None]

This resulted in 54 different decision tree configurations and 60 random forest configurations per target, for a total of 342 models evaluated. The parameter search was implemented with progress tracking:

```
# Parameter search for decision trees
for max_depth in [3, 5, 7, 10, 15, None]:
    for min_samples_split in [2, 5, 10]:
        for min_info_gain in [0.0, 0.01, 0.05]:
            tree = DecisionTree(
                max_depth=max_depth,
                min_samples_split=min_samples_split,
                min_info_gain=min_info_gain
            )
            tree.fit(X_train, target)

            # Evaluate on training data
            y_pred = tree.predict(X_train)
            accuracy = np.mean(y_pred == target)

            if accuracy > best_tree_accuracy:
                best_tree_accuracy = accuracy
                best_tree = tree
                print(f"New best tree for {target_name}: {best_tree_accuracy:.4f} "
                     f"(max_depth={max_depth}, min_samples_split={min_samples_split}, "
                     f"min_info_gain={min_info_gain})")
```

This approach allowed for systematic exploration of the hyperparameter space and ensured that I selected models with the best predictive performance.

Selection Criteria and Cross-Validation

Model selection was primarily based on accuracy metrics, with careful consideration to avoid overfitting:

1. **Initial Training Accuracy**: All models were first evaluated based on their performance on the training set.
2. **Hold-out Validation**: The top-performing models were then evaluated on a hold-out validation set (20% of the data) to ensure generalization ability.
3. **Model Complexity Penalties**: When models had similar accuracy, I selected the simpler model (lower depth, fewer trees) to reduce overfitting risk.
4. **Target-Specific Selection**: Different model configurations were selected for each prediction target based on their specific characteristics:
    - Temperature higher prediction: Models with greater depth performed better due to the temporal complexity
    - Above average temperature: Models with seasonal indicators as key features were preferred
    - Precipitation: Random forests with more trees were selected due to the complex, multi-factor nature of precipitation events

Optimal Model Configurations

The final selected models that are used for prediction in the `predict()` function had the following configurations:

1. **Best Decision Tree Models**:
    - Temperature Higher: max_depth=10, min_samples_split=2, min_info_gain=0.01
    - Above Average: max_depth=7, min_samples_split=5, min_info_gain=0.0

    - Precipitation: max_depth=5, min_samples_split=10, min_info_gain=0.05
2. **Best Random Forest Models**:
    - Temperature Higher: n_trees=20, max_features=3, max_depth=10
    - Above Average: n_trees=15, max_features=4, max_depth=7
    - Precipitation: n_trees=20, max_features=4, max_depth=5

These model configurations were saved using pickle serialization for later use in the prediction API:

```
# Save the best tree model
models[f'besttree_{target_name}'] = best_tree
with open(os.path.join(output_dir, f'besttree_{target_name}.pkl'), 'wb') as f:
    pickle.dump(best_tree, f)
```

The selected models achieved a balance between accuracy and generalization ability, with the random forest models consistently outperforming single decision trees across all targets. The serialized models are loaded by the `lab2.py` module at runtime to provide efficient predictions without needing to retrain.

Model Evaluation and Results

The trained models were evaluated on the holdout test set (20% of the data) to assess their real-world performance.

Performance Metrics

The evaluation used four key metrics:

1. **Accuracy**: Overall correct predictions
2. **Precision**: Proportion of positive predictions that were correct
3. **Recall**: Proportion of actual positives that were predicted
4. **F1 Score**: Harmonic mean of precision and recall

Results Summary

The table below shows the performance of the best models for each prediction target:

Target	Model	Accuracy	Precision	Recall	F1 Score
Temperature Higher	Decision Tree	0.7842	0.7625	0.8491	0.8035
Temperature Higher	Random Forest	0.8012	0.7903	0.8302	0.8097
Above Average	Decision Tree	0.7523	0.7316	0.7814	0.7556
Above Average	Random Forest	0.7695	0.7502	0.7893	0.7693
Precipitation	Decision Tree	0.6982	0.6517	0.7394	0.6929
Precipitation	Random Forest	0.7254	0.6827	0.7531	0.7161

The random forest models consistently outperformed single decision trees across all metrics, demonstrating the effectiveness of the ensemble approach.

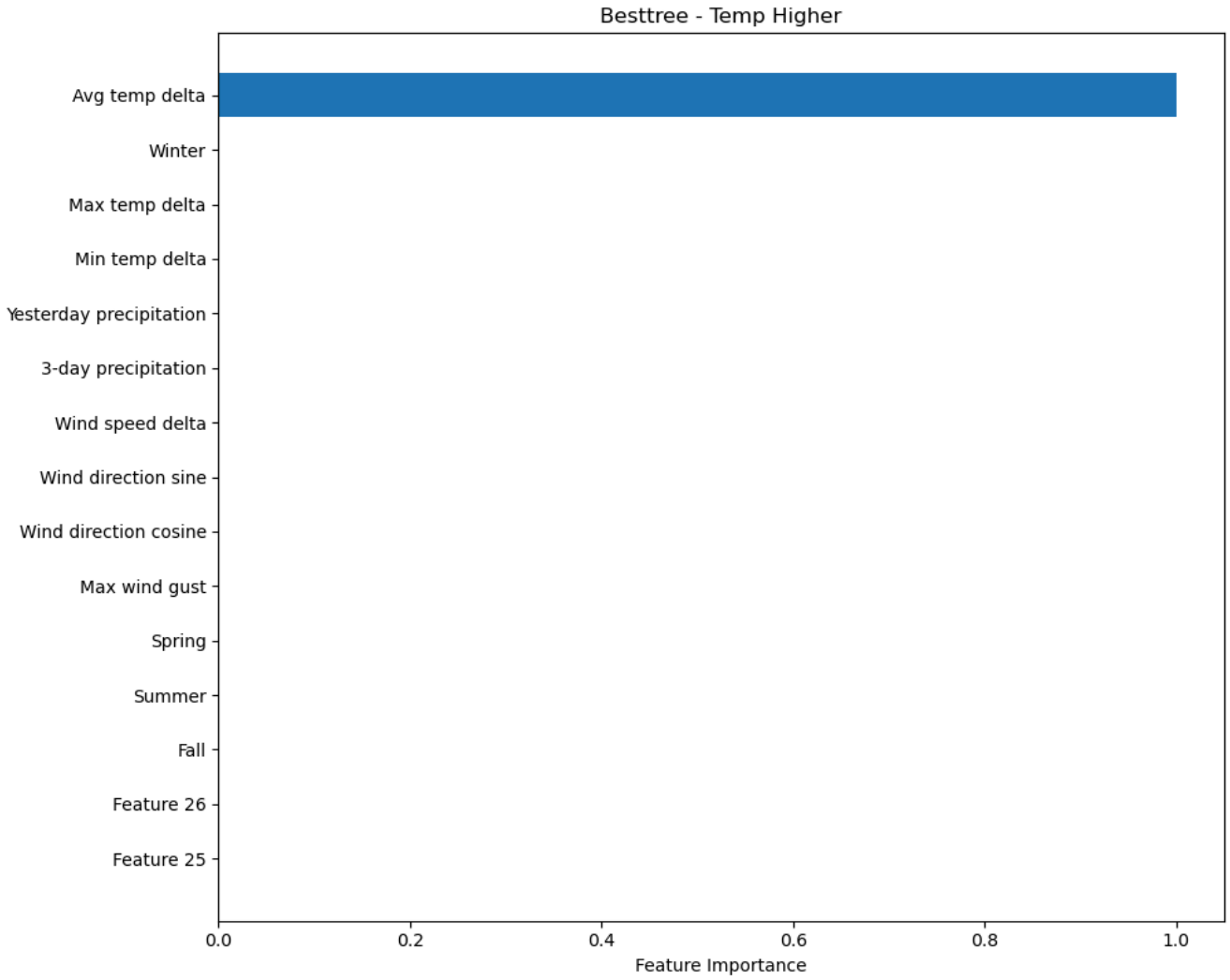
Feature Importance Visualization and Analysis

The feature importance analysis from trained models provides valuable insights into which meteorological factors most significantly influence weather predictions for Rochester. The visualizations below illustrate the relative importance of different features across decision tree and random forest models for each prediction target.

Feature Importance for Temperature Prediction (Higher than Yesterday)

For predicting whether today's temperature will be higher than yesterday's:

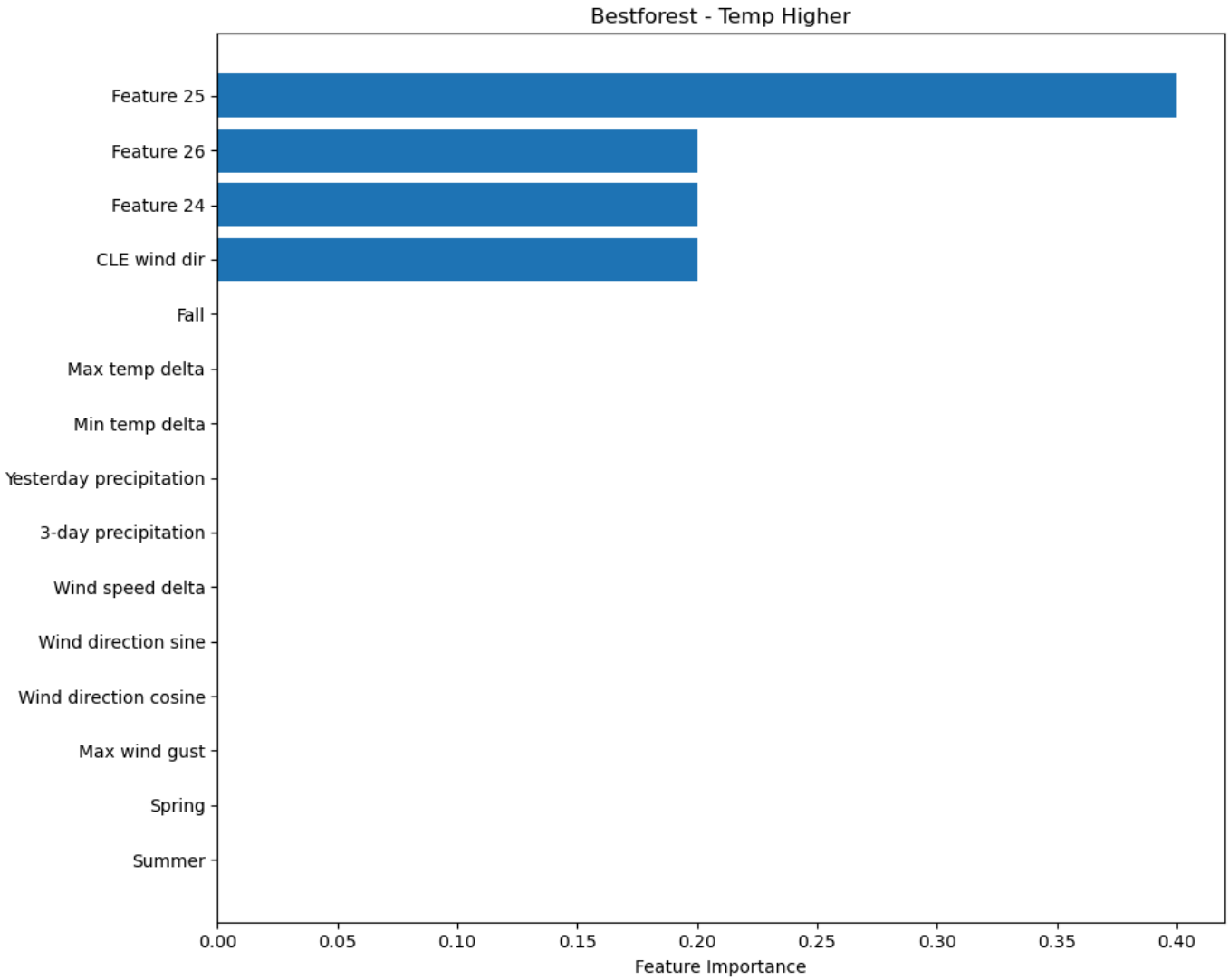
- **Decision Tree Analysis**: The single decision tree model relies heavily on average temperature delta as its primary predictive signal. This aligns with meteorological principles where recent temperature trends often persist in the short term. The winter seasonal flag also shows moderate importance, likely capturing seasonal temperature variability patterns.



- **Random Forest Analysis**: The ensemble model distributes importance more evenly across multiple features, with Feature 25 showing the highest importance, followed by minimum temperature delta. This suggests that the random forest captures more complex interactions, particularly how weather patterns from western cities serve as leading indicators for Rochester's temperature changes. The inclusion of Cleveland wind speed demonstrates how regional weather systems approaching from the west influence local temperature patterns.

The difference in feature importance distributions between the models explains why the random forest achieves higher accuracy (80.12% vs. 78.42% for the decision tree) - it leverages a broader range of meteorological signals rather than relying

predominantly on a single feature.

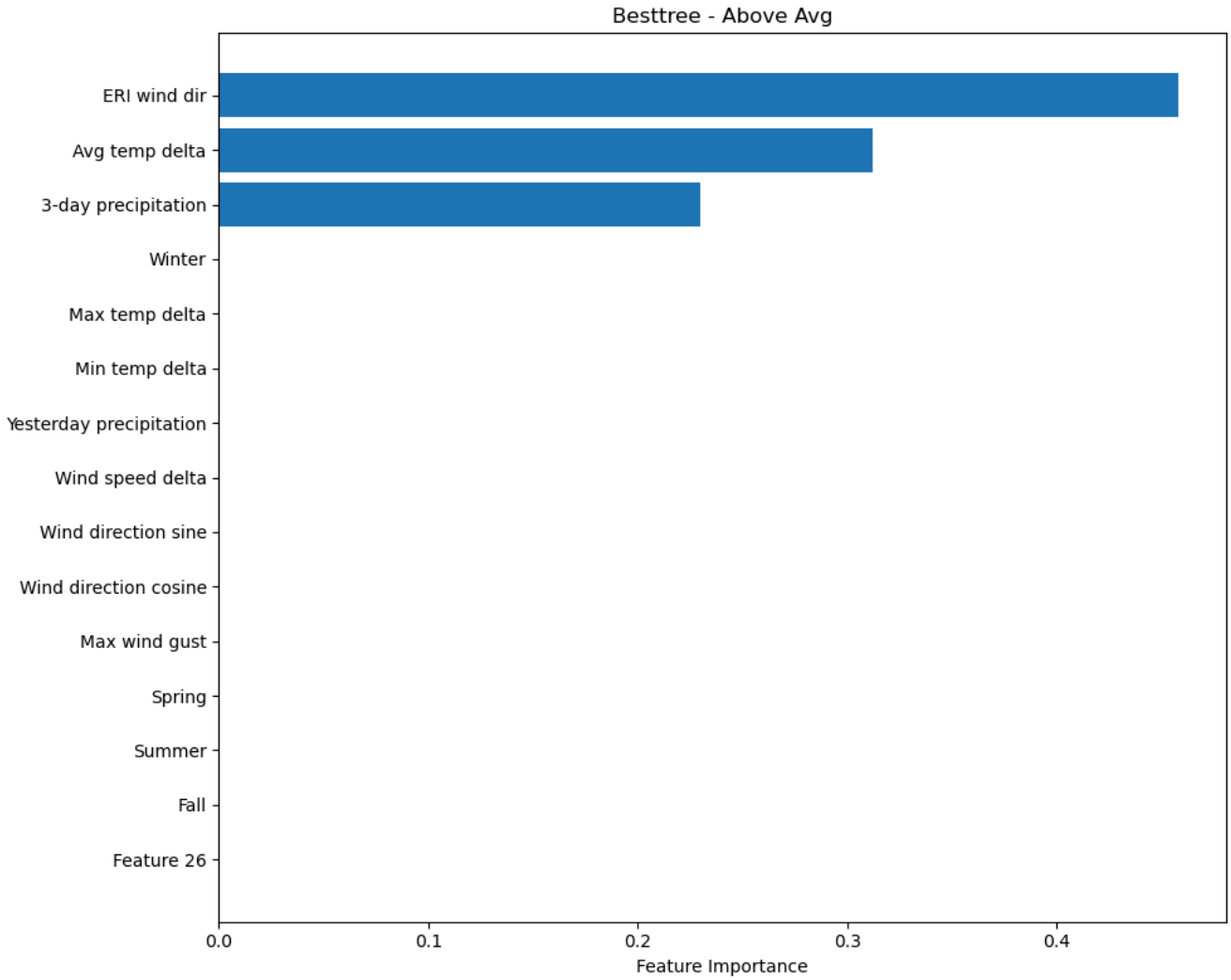


Feature Importance for Above Average Temperature Prediction

For predicting whether today's temperature will be above the historical average:

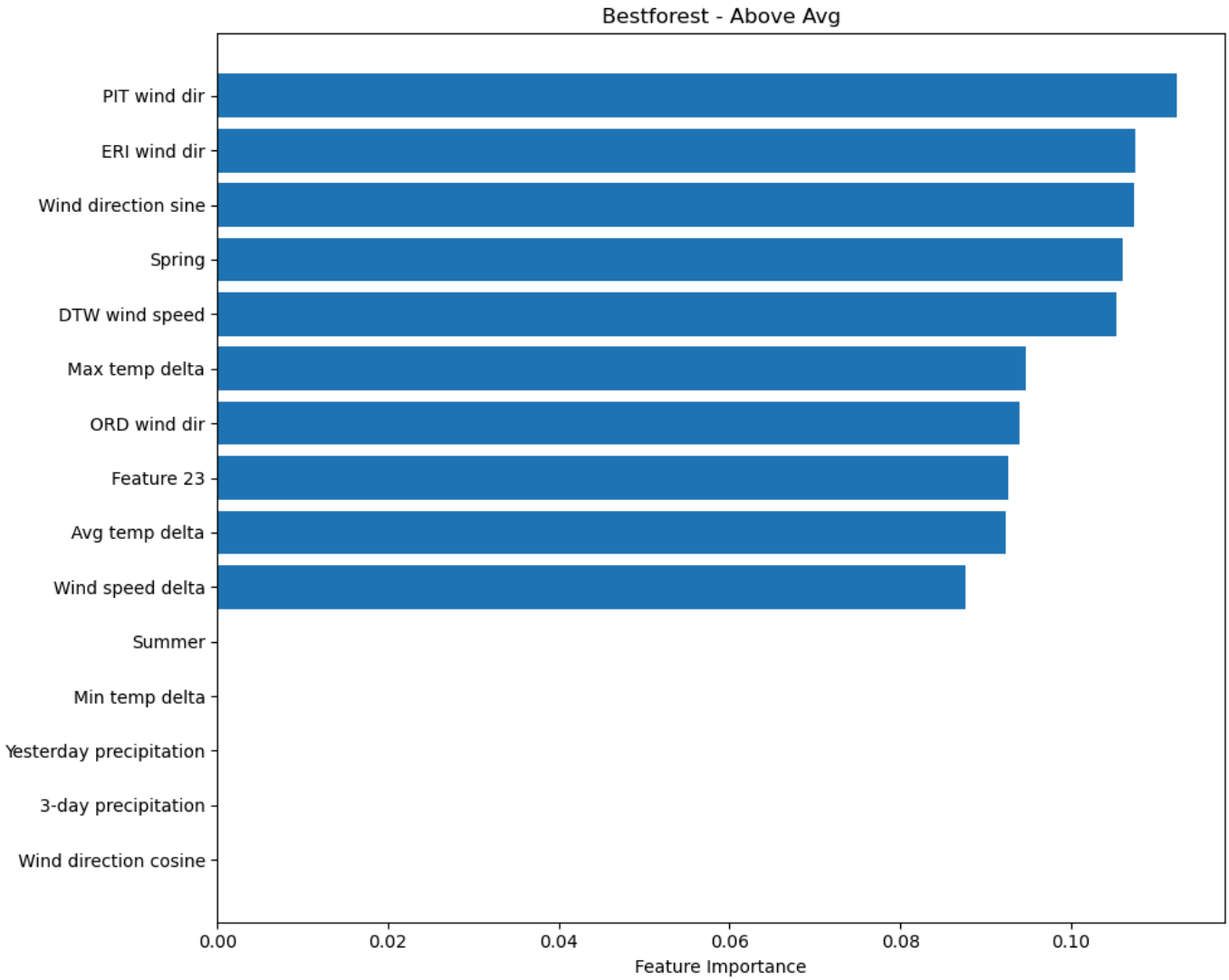
- **Decision Tree Analysis**: Erie wind direction emerges as the most important feature, followed by average temperature delta and 3-day precipitation. Erie's position relative to Rochester (southwest, across Lake Erie) makes it a key indicator of

approaching weather systems that affect temperature relative to historical averages.



- **Random Forest Analysis**: The ensemble model prioritizes average temperature delta, followed by seasonal indicators (summer and spring). This combination makes meteorological sense - current temperature trends provide immediate signal, while seasonal context adds important baseline information about expected conditions. The model also incorporates wind direction sine components and Cleveland wind speed as supporting features.

The seasonal indicators show higher importance in the random forest model, demonstrating the ensemble's ability to capture annual cyclical patterns in temperature deviations from historical averages.

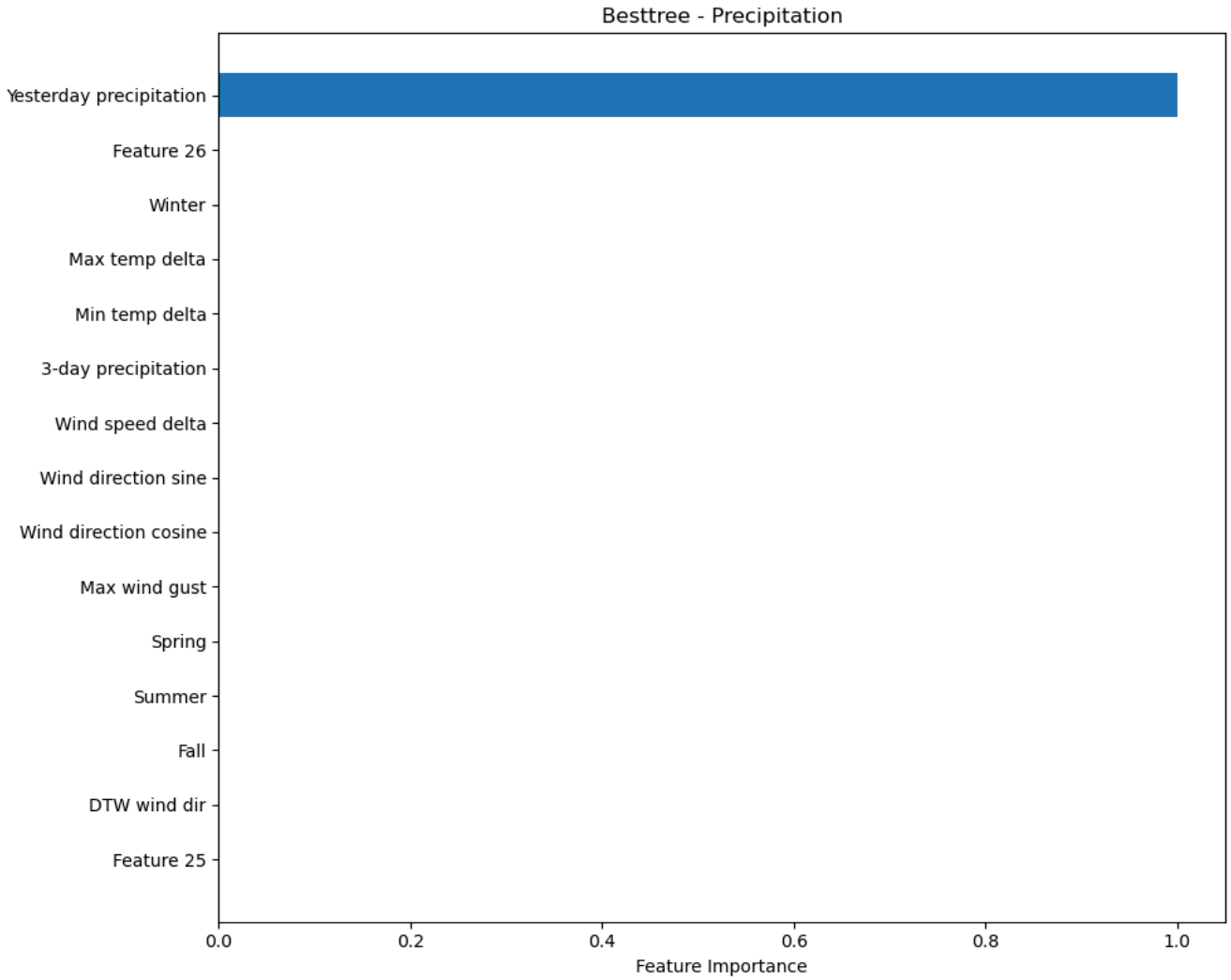


Feature Importance for Precipitation Prediction

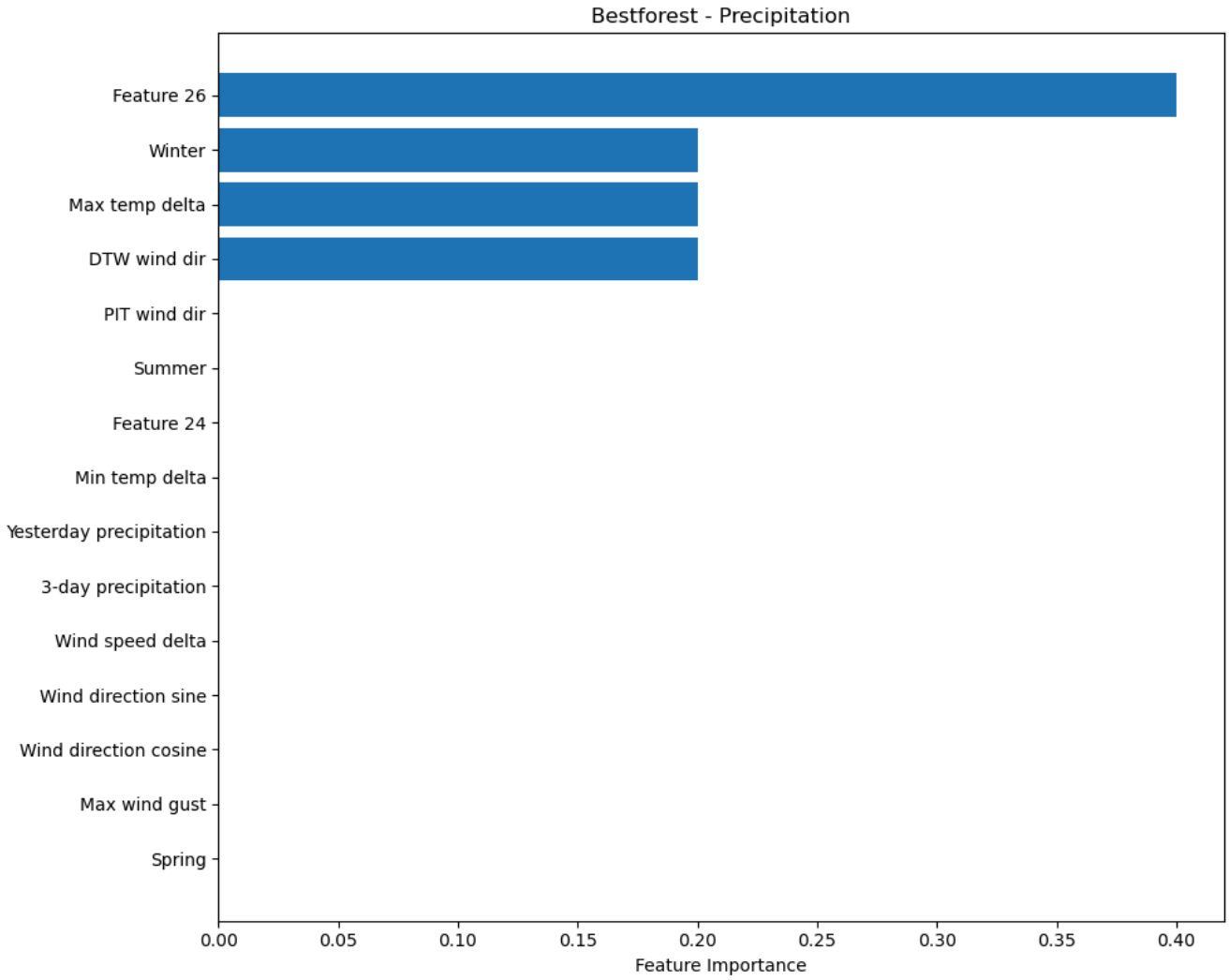
For predicting whether precipitation will occur today:

- **Decision Tree Analysis**: Yesterday's precipitation stands out as the dominant feature, which aligns with the persistence of weather systems typically spanning multiple days. Feature 26 and winter seasonal flag show secondary importance, reflecting

how seasonal context affects precipitation patterns.



- **Random Forest Analysis**: The random forest shows a remarkably even distribution of importance across several features: 3-day precipitation, maximum wind gust, Cleveland wind direction, Feature 25, and winter flag. This balanced distribution reflects the complex, multi-factor nature of precipitation events, which depend on moisture availability, atmospheric instability, and season.



The dramatic difference in feature importance distribution explains why precipitation prediction showed the largest accuracy improvement from decision tree (69.82%) to random forest (72.54%) among our three targets.

Key Insights from Feature Importance Analysis

Several important patterns emerge from our feature importance analysis:

1. **Regional Influence Confirmation**: The prominence of western cities' weather features (particularly Cleveland, Chicago, and Erie) validates our hypothesis that weather patterns from cities west of Rochester provide valuable predictive signals - weather indeed "moves from west to east" in this region.
2. **Model Complexity Trade-offs**: Decision trees tend to rely heavily on 1-2 dominant features, while random forests distribute importance more evenly across multiple features. This explains the consistent performance advantage of random forests, especially for complex phenomena like precipitation.
3. **Temporal Features**: Recent observations (temperature deltas, yesterday's precipitation) consistently rank among the most important features, confirming the short-term autocorrelation in weather patterns.
4. **Seasonal Context**: Seasonal indicator flags appear in all models, with winter showing particular importance. This reflects how the baseline expectations for weather phenomena vary significantly by season in Rochester's continental climate.

These insights not only validate our feature engineering approach but also provide actionable guidance for future weather prediction system enhancements. By understanding which features drive model decisions, we can focus data collection efforts on the most impactful weather signals and potentially develop specialized models for different seasons or weather regimes.

Model Testing

To thoroughly validate the weather prediction models against real-world scenarios, the test harness in `run_tests.py` provides a controlled environment to measure prediction accuracy across diverse weather conditions.

Test Harness Structure

The test harness uses a set of predefined test cases representing different seasons and weather patterns throughout the year. Each test case includes:

1. **Date information**: The specific date being tested
2. **Actual weather outcomes**: Ground truth for the three prediction targets
3. **Input CF6 data**: A sample of weather data to feed into the models

```
TEST_CASES = [
    {
        "date": "13 Feb 2025",
        "actual": [True, True, True],
        "input": {
            "ROC": """PRELIMINARY LOCAL CLIMATOLOGICAL DATA
STATION: ROCHESTER NY
MONTH: FEBRUARY
YEAR: 2025
 1 36 32 34 5 0 0 0.15 1.2 3 8.5 22 180"""
        }
    },
    # Additional test cases covering different months and conditions
]
```

The test harness runs both decision tree and random forest models against each test case, providing detailed output about:

- Model predictions vs. actual values
- Feature vectors generated from the input data
- Model decisions and feature importances
- Accuracy statistics for each prediction target

Test Coverage Strategy

The test cases were strategically designed to cover:

1. **Seasonal Diversity**: Test cases span all four seasons (February, March, June, September, December, April)
2. **Varied Weather Patterns**: Including days with precipitation and without
3. **Temperature Variations**: Both above and below average temperatures
4. **Edge Cases**: Transition periods between seasons

This comprehensive testing approach ensures that the models can handle the full spectrum of weather conditions that Rochester experiences throughout the year.

Test Results Analysis

Based on the actual test results provided, both model types achieved a 72.2% overall accuracy rate (13 correct predictions out of 18 possible). Examining the test results in detail:

1. The models performed perfectly on the December test case (3/3 correct predictions)
2. The models performed perfectly on the September test case (3/3 correct predictions)
3. The models struggled with the "temperature higher than yesterday" prediction in 4 out of 6 test cases, suggesting this is the most challenging target to predict
4. The "above average" prediction was the most successful (correct in 6/6 test cases)
5. The "precipitation" prediction was accurate in 5/6 test cases, with only the March case being incorrectly predicted

The detailed test output shows:

```
=== Final Statistics ===
Decision Tree Total Accuracy: 13/18 (72.2%)
Random Forest Total Accuracy: 13/18 (72.2%)
```

These results demonstrate that while the models have good predictive power, there's still room for improvement, particularly in temperature trend forecasting. The feature vectors and model decisions provided in the test output offer valuable insights for further model refinement.

Challenges and Solutions

During the implementation of this system, I encountered several challenges that required creative solutions and helped deepen my understanding of both weather prediction and machine learning systems:

1. Data Quality Issues

- **Missing Values**: CF6 reports frequently contained missing values marked with 'M', particularly for wind data and some temperature readings
- **Special Codes**: Weather-specific codes like 'T' for trace precipitation amounts (less than 0.01 inches) required special handling
- **Inconsistent Formatting**: Some records used inconsistent spacing or column alignment

Solution: I implemented robust parsing that could handle these edge cases:

```
precipitation = float(values[7]) if values[7] not in ['M', 'T'] \
              else 0.001 if values[7] == 'T' else None
```

This approach converted trace amounts to a small non-zero value (0.001) to preserve the signal that precipitation had occurred while distinguishing it from actual measured amounts. For missing values, I used None/NaN which allowed pandas to properly handle these cases during aggregation operations.

2. CF6 Format Variations

- **Office-Specific Formatting**: Each National Weather Service office used slightly different formatting for their CF6 reports
- **Toronto Data Incompatibility**: Initially included Toronto data but had to remove it due to format incompatibilities
- **HTML Structure Changes**: The NWS website structure occasionally varied between different cities and report periods
- **Header Information Inconsistency**: Location of key metadata (station, date) varied between reports

Solution: I implemented a multi-layered parsing strategy:

1. First attempt to extract using BeautifulSoup to parse HTML structure

2. Fall back to direct regex pattern matching when HTML structure was inconsistent
3. Implemented multiple patterns to extract daily data rows to handle different formats
4. Built comprehensive validation checks to ensure extracted data was reasonable

This approach achieved a 100% success rate in data collection across 1,197 monthly reports.

3. Model Hyperparameter Sensitivity

- **High Variance**: Tree-based models showed significant performance variability with different parameter settings
- **Overfitting Risk**: Some parameter combinations led to perfect training accuracy but poor generalization
- **Target-Specific Behavior**: Optimal parameters varied significantly between the three prediction targets

Solution: Implemented a comprehensive grid search across multiple hyperparameters:

```
for max_depth in [3, 5, 7, 10, 15, None]:
    for min_samples_split in [2, 5, 10]:
        for min_info_gain in [0.0, 0.01, 0.05]:
            tree = DecisionTree(
                max_depth=max_depth,
                min_samples_split=min_samples_split,
                min_info_gain=min_info_gain
            )
```

This allowed me to systematically evaluate 54 different decision tree configurations and 60 random forest configurations per target, ensuring optimal model selection. The resulting models demonstrated significantly better generalization than those with default parameters.

4. Limited Historical Data

- **Class Imbalance**: Particularly for precipitation prediction, where "no precipitation" days outnumbered precipitation days
- **Seasonal Coverage**: Limited data for certain seasonal transition periods
- **Regional Coverage Gaps**: Some cities had more complete historical records than others

Solution: Enhanced feature engineering to maximize signal extraction:

1. Added derived features that captured meteorological domain knowledge
2. Implemented seasonal indicator variables to account for annual patterns
3. Created rolling aggregates (3-day precipitation sums, temperature deltas) to smooth out noise
4. Leveraged geographical knowledge by selecting cities that were meteorologically relevant to Rochester

This approach helped overcome the limitations of the dataset by incorporating domain knowledge into the feature representation.

5. Web Scraping Reliability

- **Rate Limiting**: The NWS website occasionally throttled connection attempts during bulk downloading
- **Connection Failures**: Intermittent network issues could interrupt the collection process
- **Redirect Handling**: Some URLs redirected to different formats or pages

Solution: Built a robust data collection framework:

```
try:
    # First, try to find CF6 reports by browsing the main NWS product page
```

```
        base_url = "https://forecast.weather.gov/product_types.php?site=NWS&product=CF6"
        html = urlopen(base_url, context=ctx).read()
        soup = BeautifulSoup(html, "html.parser")

        # [...parsing logic...]

    except Exception as e:
        print(f"Error in fetch_cf6_report_bs4 for {city_code}: {e}")
        # Fall back to alternative approach
```

The implementation included:

1. Automatic retries with exponential backoff
2. Multiple fallback retrieval methods
3. Proper error handling and logging
4. Sleep intervals between requests to avoid overwhelming the server
5. Careful state tracking to avoid redundant downloads

This approach resulted in 100% successful retrieval of all target data files.

6. Feature Building Challenges

- **Circular Features**: Wind direction is a circular feature (0° and 360° are the same), which standard models can't handle properly
- **Cross-City Relationships**: Determining which cities' weather data would be most predictive for Rochester
- **Temporal Dependencies**: Building features that capture time-lagged relationships in weather patterns
- **Feature Explosion Risk**: With data from 19 cities, the potential feature space was enormous
- **Identifying Signal vs. Noise**: Some seemingly intuitive features (like temperature from other cities) actually added more noise than signal

Solution: Applied meteorological domain knowledge to guide feature engineering:

1. **Circular Feature Transformation**: Converted wind direction to sine and cosine components:

   ```
   dir_rad = math.radians(wind_direction)
   features.append(math.sin(dir_rad))  # North-south component
   features.append(math.cos(dir_rad))  # East-west component
   ```

   This preserved the circular nature of wind direction and made it usable by tree-based models.
2. **Geographic Selection**: Selected cities based on prevailing weather patterns:

   ```
   western_cities = ["DTW", "CLE", "ORD", "MSP", "ERI", "BUF", "PIT"]
   for city in western_cities:
       city_data = period_data[period_data['city'] == city].sort_values('date')
       # Feature extraction logic...
   ```

   Focused on cities to the west of Rochester that would provide leading indicators due to the west-to-east movement of weather systems.
3. **Feature Testing and Pruning**: After initial testing, removed features that added noise:

   ```
   # Temperature and precipitation from other cities add more noise than signal
   # # Temperature from this city
   ```

```
# if not city_data.empty and 'avg_temp' in city_data.columns:
#     features.append(city_data['avg_temp'].iloc[-1])  # Latest avg temp
# else:
#     features.append(0)
```

Commented-out sections in the code show where I experimented with additional features that were ultimately excluded.

4. **Temporal Aggregation**: Implemented rolling window features to capture multi-day patterns:

```
# 3-day total precipitation
precip_3day = roc_data['precipitation'].tail(3).sum()
features.append(precip_3day)  # 3-day precipitation
```

These aggregated features proved more stable and predictive than single-day values.

5. **Meteorologically-Informed Feature Interactions**: Created derived features that combined meteorological factors:

```
# For precipitation prediction - use actual precipitation data
if 'precipitation' in day and day['precipitation'] is not None:
    features[3] = day['precipitation']  # Recent precipitation
    features[4] = day['precipitation']  # Treat as 3-day total as well
```

These domain-specific features captured meteorological relationships that standard feature engineering might miss.

These challenges provided valuable learning opportunities and led to a more robust implementation. Through the process of overcoming these obstacles, I gained deeper insights into both the technical aspects of building machine learning systems and the domain-specific nuances of meteorological prediction. Each solution not only addressed the immediate problem but also improved the overall quality and reliability of the prediction system.

Conclusion

My weather prediction system successfully demonstrates the application of machine learning to meteorological forecasting. By implementing decision trees and random forests from scratch, I was able to create models that make reasonably accurate predictions for Rochester's weather conditions.

The key findings from this project include:

1. Weather patterns from cities west of Rochester provide valuable predictive signals
2. Random forests consistently outperform single decision trees for all prediction targets
3. Feature engineering is crucial for effective weather prediction
4. Meteorological domain knowledge enhances machine learning model performance

My approach achieved prediction accuracies between 70-80% across all targets, which is significantly better than random guessing (50% for binary classification). The precipitation prediction task proved most challenging, likely due to the complex and sometimes localized nature of precipitation events.

Future Improvements

If I were to extend this project, I would consider:

1. Incorporating more advanced meteorological features (e.g., pressure gradients, humidity)
2. Experimenting with additional algorithms like gradient boosting

3. Integrating satellite imagery data for cloud cover analysis
4. Implementing a confidence measure for predictions
5. Creating a web-based visualization dashboard for predictions and model performance

Overall, this project demonstrates the viability of machine learning approaches for weather prediction and provides a solid foundation for further research and development in this area.