CONFORMAL PREDICTION

Michael Ghattas

Mike Worden

TOPICS

Motivations

- *More efficient approach than Bayesian Estimation*
- *Learn from/Experiment with Valeriy Manokhin found on Social Media*

Overview of Bayesian Estimation

- *Introduction to Bayesian Estimation*
- *From Bayesian Estimation to Conformal Prediction*
- *Bayesian must go through prior to provide distribution*

Overview of Conformal Prediction

- *Nonconformity Scores*
- *Algorithmic Steps*
- *Complexity Analysis (compare using sentiment analysis)*
- *Applications of Conformal Prediction (Show CODE snippet)*

Application Demo

Conclusion/Next steps

MOTIVATIONS

Both of us have motivations to find efficient classification algorithms

- *Michael G – Works in data science*
- *Mike W – Interested in applications of classifiers in cybersecurity logs analysis*

Both interested in finding an improvement over Bayesian methods

Learned of conformal prediction through interaction with Valeriy Manokhin on Social Media. (LinkedIn)

- *Reviewed his papers & publications, including*

  Practical Guide to Applied Conformal Prediction in Python: Learn and apply the best uncertainty frameworks to your industry applications. (ISBN: 1805122762)

Conformal Prediction was a suitable topic for investigation both for education, but also our existing careers.

Valeriy Manokhin, PhD, MBA, CQF

OVERVIEW OF BAYESIAN ESTIMATION

Bayes' Theorem - **The probability of event A, given that event B has occurred:**

$$P(A|B) = \frac{P(A) \cdot P(B|A)}{[P(A) \cdot P(B|A)] + [P(\bar{A}) \cdot P(B|\bar{A})]}$$

Key Concept - **Updates prior beliefs based on new evidence.**

Key Terms:

- **Posterior Distribution:** Bayesian inference relies on updating beliefs through the posterior distribution, which combines the likelihood of the observed data with the prior distribution
- **Credible Intervals:** Unlike frequentist confidence intervals, Bayesian credible intervals offer a probability-based interpretation of parameter uncertainty

PROS/CONS OF BAYESIAN ESTIMATION

Advantages of Bayesian Methods

- *Probabilistic framework for inference.*
- *Intuitive interpretation of uncertainty.*
- *Flexibility in model updating*

Critiques of Bayesian Methods

- ***Subjectivity in Prior Selection:*** *Researcher intuition & common heuristics (e.g., principle of indifference).*
- ***Computational Complexity:*** *High-dimensional models require costly calculations.*
  - *Example: Use of Markov Chain Monte Carlo (MCMC) used to explore priors.*
- ***Scalability Issues:*** *High-dimensional Bayesian models struggle to scale efficiently for big data.*

ALTERNATIVE – CONFORMAL PREDICTION (CP)

•**Key Idea:** Constructs prediction sets without requiring a full probability distribution,

- *Computes **nonconformity scores** to measure deviations.*
- *Uses empirical quantiles to form **prediction sets**.*

Feature	Bayesian Inference	Conformal Prediction
Prior Knowledge	Required	Not Needed
Uncertainty Estimation	Posterior Distributions	Prediction Sets
Computational Cost	H (MCMC)	Efficient (Quartile Based)
Scalability	Limited for Big Data	Highly Scalable

APPLICATIONS OF CONFORMAL PREDICTION

- **Healthcare:** Medical diagnostics with uncertainty estimation
- **Finance:** Stock market risk assessment
- **Cybersecurity:** Anomaly detection in network traffic
- **Our Project:** Forecasting news article counts across:
  - Entertainment, Politics, Sports, Technology

APPROACH AND COMPUTATIONAL COMPLEXITY

1. Train a base model (linear regression).
2. Compute residuals (nonconformity scores).
3. Estimate threshold using empirical quantiles.
4. Construct prediction intervals.

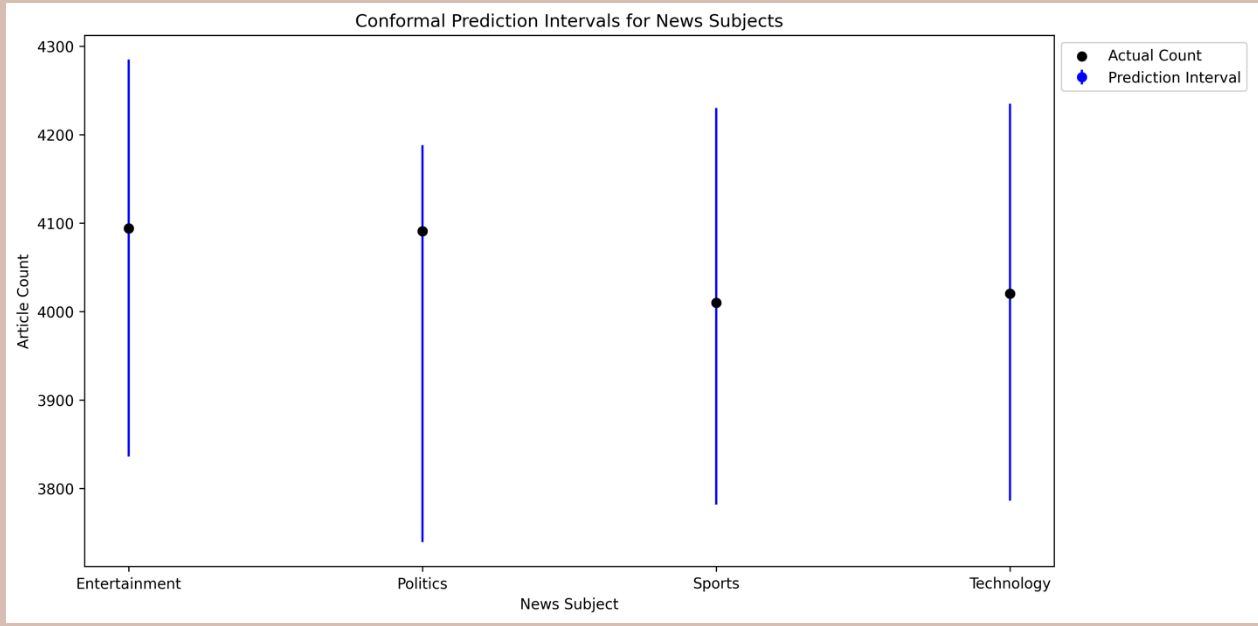
CP Method	Complexity	Pros	Cons
Inductive CP (ICP)	O(n)	Fast, Scalable	Wider Intervals
Transductive CP (TCP)	O(nk)	Tighter Intervals	Computationally Expensive
Mondrian CP	O(n log n)	Efficient & adaptive	Assumes conditional independence

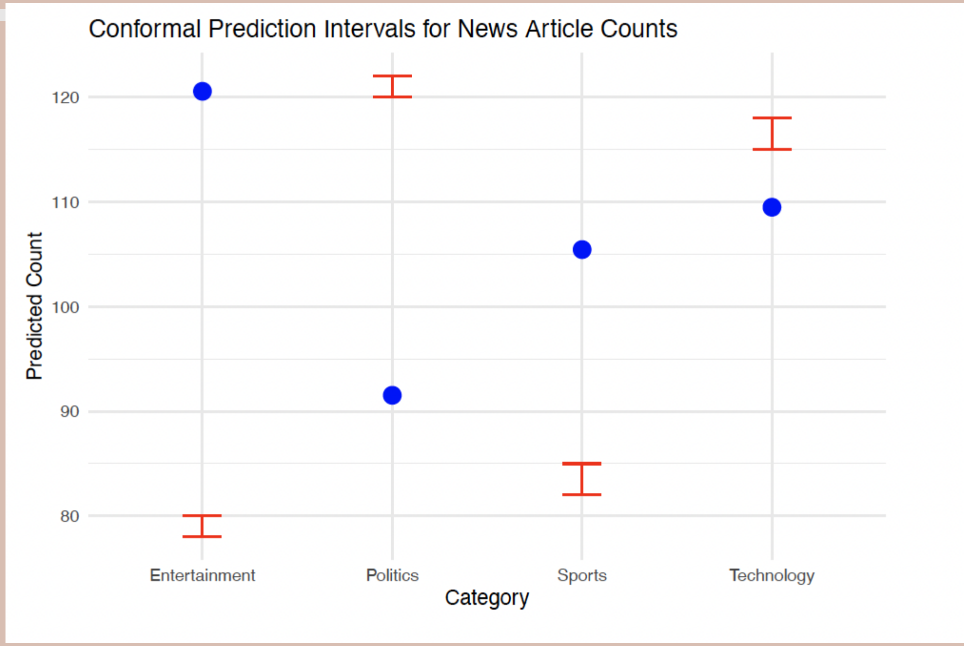
EXAMPLE: CLASSIFICATION OF NEWS STORIES

```python
> class ConformalCoverForest:…

# --- Generate Synthetic Data ---
np.random.seed(42)
subjects = ["Health", "Entertainment", "Sports", "Politics", "Technology"]
num_samples = 100

data = pd.DataFrame({
    "Subject": np.random.choice(subjects, num_samples),
    "Feature1": np.random.randn(num_samples) * 100 + 500,
    "Feature2": np.random.randn(num_samples) * 50 + 200,
    "Actual_Count": np.random.randint(3900, 4100, num_samples)
})

# --- Train Model ---
X = data.drop(columns=["Actual_Count", "Subject"])
y = data["Actual_Count"]
model = ConformalCoverForest(alpha=0.05)
model.fit(X, y)

# --- Predict and Evaluate ---
X_test = X[:10]
lower, upper = model.predict(X_test)

# --- Save Output CSV ---
output_df = pd.DataFrame({
    "Subject": data["Subject"][:10],
    "Actual_Count": y[:10],
    "Lower_Bound": lower,
    "Upper_Bound": upper,
    "Within": ((y[:10] >= lower) & (y[:10] <= upper)).astype(int)  # One-hot encoding for correctness
})
output_csv = "COMP_4581_Project_Output.csv"
output_df.to_csv(output_csv, index=False)
print(f"Output saved to {output_csv}")

# --- Save Log File ---
log_file = "COMP_4581_Project_Log.txt"
with open(log_file, "w") as log:
    log.write(f"Model Execution Summary:\n")
    log.write(f"Training Time: {model.train_time:.4f} seconds\n")
    log.write(f"Prediction Time: {model.predict_time:.4f} seconds\n")
    log.write(f"Total Run Time: {model.train_time + model.predict_time:.4f} seconds\n")

print(f"Log saved to {log_file}")
```

```python
# --- Generate and Save Plot, Aggregate Data, and Ensure One Entry Per Category ---
output_df_grouped = output_df.groupby("Subject", as_index=False).mean()

plt.figure(figsize=(12, 6))
categories = output_df_grouped["Subject"]
actual_counts = output_df_grouped["Actual_Count"]
lower_bounds = output_df_grouped["Lower_Bound"]
upper_bounds = output_df_grouped["Upper_Bound"]

# Compute error bar range
error_bars = [actual_counts - lower_bounds, upper_bounds - actual_counts]

# Scatter plot for actual counts
plt.scatter(categories, actual_counts, color="black", label="Actual Count", zorder=3)

# Error bars for prediction intervals
plt.errorbar(categories, actual_counts, yerr=error_bars, fmt="o", color="blue", label="Prediction Interval")

plt.xlabel("News Subject")
plt.ylabel("Article Count")
plt.title("Conformal Prediction Intervals for News Subjects")

# Move legend outside the plot
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))

# Adjust layout and save
plt.tight_layout()
plt.savefig("COMP_4581_Project_Plot.png", bbox_inches="tight", dpi=300)

print("Plot saved as COMP_4581_Project_Plot.png")



##
# Note:
# Completion, debugging, and validation of the code was assisted by GenAI/LLMs.
##
```

EXAMPLE: CLASSIFICATION OF NEWS STORIES






```
Model Execution Summary:
Training Time: 0.1065 seconds
Prediction Time: 0.0063 seconds
Total Run Time: 0.1129 seconds
```

CONCLUSION/NEXT STEPS

- ***Conclusion***
  - *Conformal Prediction (CP) ensures reliable prediction sets with finite-sample validity*
  - *Advancements have made CP feasible for large-scale applications despite computational challenges.*
  - *See github repo for practical example*

- **Future Considerations:**
  - *Improving Interval Tightness: Adaptive nonconformity scores.*
  - *Comparing CP with Bayesian Methods: Performance trade-offs.*
  - *Scaling CP for Big Data: Efficient parallel processing.*

THANK YOU

Michael Ghattas

michael.Ghattas@du.edu

Mike Worden

mike.worden@du.edu

CLASS DEFINITION

```python
class ConformalCoverForest:
    """
    Conformal Prediction using Random Forest for classification-based uncertainty estimation.
    This version improves residual estimation to prevent overfitting and enhances generalization.

    Attributes:
    - alpha: Significance level (1 - confidence level).
    - base_model: The base regression model (default is RandomForestRegressor).
    - residuals: Stores the absolute residuals from calibration.
    - train_time: Tracks model training time.
    - predict_time: Tracks prediction interval computation time.
    - label_encoder: Encodes categorical target labels.
    """

    def __init__(self, alpha=0.05, base_model=None):
        """
        Initializes the Conformal Cover Forest model with a specified confidence level.

        Parameters:
        - alpha (float): Significance level (e.g., 0.05 means 95% confidence).
        - base_model: The base regression model (default: RandomForestRegressor with better hyperparameters).
        """
        self.alpha = alpha
        self.base_model = base_model if base_model else RandomForestRegressor(
            n_estimators=300, max_depth=10, bootstrap=True, random_state=42
        )
        self.residuals = None
        self.train_time = 0.0
        self.predict_time = 0.0
        self.is_fitted = False  # Track if model is fitted
        self.label_encoder = None  # For encoding categorical target labels


    def preprocess_data(self, X, y):
        """
        Converts categorical features and target labels into numeric format.

        Parameters:
        - X (DataFrame or array-like): Feature matrix (may contain categorical columns).
        - y (Series or array-like): Target labels (categorical or numerical).

        Returns:
        - X_processed: Numeric feature matrix.
        - y_processed: Encoded numeric target variable.
        """

        X = pd.DataFrame(X)  # Ensure X is a DataFrame

        # Convert categorical features to numerical (One-Hot Encoding)
        for col in X.select_dtypes(include=['object']).columns:
            one_hot = pd.get_dummies(X[col], prefix=col)
            X = X.drop(col, axis=1)
            X = pd.concat([X, one_hot], axis=1)

        # Encode target labels if they are categorical
        if isinstance(y, pd.Series) and y.dtype == 'object':
            self.label_encoder = LabelEncoder()
            y = self.label_encoder.fit_transform(y)

        return X.values, y
```

```python
    def fit(self, X, y):
        """
        Trains the model on provided data and calculates residuals for conformal prediction.

        Parameters:
        - X (array-like): Feature matrix.
        - y (array-like): Target variable (discrete class labels).

        Steps:
        1. Preprocesses categorical data into numeric format.
        2. Splits data into training (80%) and calibration (20%) sets.
        3. Fits the RandomForest model on the training set.
        4. Predicts on the calibration set and computes residuals using MAD + noise.
        """

        X, y = self.preprocess_data(X, y)  # Convert categorical data

        # Increase calibration set size for better generalization (80-20 split)
        X_train, X_calib, y_train, y_calib = train_test_split(X, y, test_size=0.2, random_state=42)

        print("Training model...")
        start_time = time.time()
        for _ in trange(1, desc="Training Progress"):
            self.base_model.fit(X_train, y_train)
        self.train_time = time.time() - start_time
        self.is_fitted = True

        print("Computing residuals on calibration set...")
        y_pred_calib = self.base_model.predict(X_calib)

        # Compute residuals using Median Absolute Deviation (MAD) and Gaussian noise
        self.residuals = np.abs(y_calib - y_pred_calib)
        mad = median_absolute_error(y_calib, y_pred_calib)

        # FIX: Reduce excess variability by scaling noise contribution
        self.residuals += mad + np.random.normal(0, mad * 0.5, size=self.residuals.shape)


    def predict(self, X_test):
        """
        Generates conformal prediction intervals.

        Parameters:
        - X_test (array-like): Feature matrix for predictions.

        Returns:
        - lower_bounds: Lower bound predictions.
        - upper_bounds: Upper bound predictions.
        """

        if not self.is_fitted:
            raise NotFittedError("Model must be fitted before predicting. Call `fit()` first.")

        print("Generating predictions...")
        X_test, _ = self.preprocess_data(X_test, np.zeros(len(X_test)))  # Encode test features

        start_time = time.time()
        y_pred = self.base_model.predict(X_test)
        q_alpha = np.quantile(self.residuals, 1 - self.alpha)

        lower_bounds = np.floor(y_pred - q_alpha)  # Rounded to discrete labels
        upper_bounds = np.ceil(y_pred + q_alpha)
        self.predict_time = time.time() - start_time

        return lower_bounds, upper_bounds
```