Uncertainty Quantification in Machine Learning Models Using Conformal Prediction

Abdulrahman Ramadan

Supervisor: Allan Peter Engsig-Karup

Technical University of Denmark

Kongens Lyngby 2023

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Contents

Chapter 1

Introduction

All models are wrong but
some are useful

George Box

Given the complexity and the massive amount of unexpected edge causes, predictive models are rarely perfect in modeling reality. Therefore, it's crucial to have techniques to help us determine the level of trust we can place in these models. The demand for understanding and managing uncertainty has been a longstanding pursuit in machine learning, especially as black-box models, which often are too complex and lack a probabilistic interpretation, are increasingly used to aid decision-making in various critical sectors like healthcare and finance. In such high-stakes environments, it's essential to understand the degree of uncertainty associated with the prediction, as it allows for more informed decision-making. For example, a doctor using a model to diagnose diseases would want to know the confidence of the model's prediction before making any critical decisions.

Achieving accurate and reliable uncertainty estimates proves to be a non-trivial task in the real world. Many commonly used methods for quantifying uncertainty, such as ensemble learning, can be computationally demanding when

applied to large-scale neural networks. The computational overhead of creating an ensemble of such models can be enormous. Furthermore, many methods rely on assumptions about the underlying distribution of data. For instance, methods like the Normal confidence interval or Student's t-confidence interval assume that the data is normally distributed. If the data deviates significantly from normality, the resulting confidence intervals may not always be valid. In many cases, using alternative methods for uncertainty quantification that are distribution-free and less resource-intensive may be more practical.

Conformal prediction is an established framework for quantifying uncertainty in machine learning models that can address the abovementioned constraints. The fundamental premise of conformal prediction is to provide prediction sets with a measure of reliability and confidence, thereby addressing the need for uncertainty quantification. Developed by Vovk, Gammerman, and Shafer in [12], the conformal prediction framework extends traditional machine learning methods to equip them with a procedure to output prediction sets associated with a certain confidence level.

In conformal prediction, we take a model as given and build a wrap-around procedure that can guarantee the future performance of the model. The procedure modifies the output of the model to produce a prediction set instead of a point prediction. More formally, assume an i.i.d dataset $\{(x_i, y_i)\}_{i=1}^{N}$ from an arbitrary distribution and an error rate $\alpha \in [0, 1]$. Under the framework, we produce a prediction set $C(x) : x \to 2^y$ that has the following property:

$$\mathbb{P}[y_{test} \in C(x_{test})] \geq 1 - \alpha \tag{1.1}$$

The advantage of conformal prediction is that the produced prediction set $C(x)$ has valid marginal coverage, as expressed in equation 1.1. In other words, the prediction sets will contain the true value with the specified confidence level. Another key property of conformal prediction is that it's distribution-free, as it does not require any assumptions about the underlying distribution of data, making it broadly applicable.

Conformal prediction is a general framework that can be applied in many contexts. Generally, the process of conformalizing a model involves the following steps:

- Calibration step: we use a separate calibration dataset to determine the range of prediction errors the model makes. These errors help to define a prediction region.
- Prediction step: the errors from the calibration step are used to produce a

set of possible values such that it satisfies the validity property expressed in equation 1.1.

This study is centered around exploring different techniques for uncertainty quantification using conformal prediction, alongside comparing their performance with similar methods. We summarize the main contributions of this study by the following:

- We explore the underlying theoretical principles of each uncertainty quantification method as well as their properties.
- We implement the methods on deep neural network models across various machine learning tasks.
- We thoroughly evaluate each method across different models and different datasets.
- We show how uncertainty estimates used with active learning can optimize a training process for a model.

The source code for implementing the methodologies is available at `https://github.com/abdualrhman/uncertainty-quantification_for_deep_learning`. Additionally, the repository provides instructions on reproducing the experimental results and generating all the figures and LaTeXtables used throughout this study.

Chapter 2

Uncertainty quantification in classification

Classification is one of the most common tasks in machine learning, where the aim is to predict the class label of a given datapoint based on its features. While the output of such a model is a class label, in many applications, it essential to quantify the uncertainty of the predicted class. The ability to express uncertainty in classification tasks can provide valuable insights into the confidence of the model's predictions.

In neural networks, particularly those used for multi-class classification tasks, the softmax function in equation 2.1 is commonly used as an activation function in the output layer.

$$\text{softmax}(x) = [\frac{e^{x_1}}{\sum_{j=1}^{L} e^{x_j}} \cdots \frac{e^{x_L}}{\sum_{j=1}^{L} e^{x_j}}]^T, \text{ where } x \in \mathbb{R}^L \tag{2.1}$$

The function maps the raw, unnormalized output of the preceding layer (logits) to a probability distribution over the target classes. Doing so ensures that the output values lie in the range [0, 1], thus resembling probabilities. One limitation of the softmax function is that it conflates the notions of model confidence and model uncertainty. A high softmax output for a particular class is often interpreted as the model's confidence in that prediction, which may not directly correspond to a low uncertainty. For instance, feeding an out-

of-training-distribution datapoint to a model, the softmax function will assign more likelihood to some classes as they all should sum op to 1.

In the subsequent sections, we explain how to overcome this limitations by using uncertainty quantification methods as an additional layer in the prediction process.

2.1 Conformal Prediction in Classification

Conformal prediction can play an essential role in classification problems due to its unique ability to provide a measure of confidence in predictions.

In this section, we introduce several methods for conformal prediction in classification and explain the underlying theory and assumptions. Furthermore, we follow the methodologies established by (Angelopoulos & Bates) in their study [1]. Their work introduces a method called Regularized Adaptive Prediction Set that conformalizes any classifier to produce smaller predictive sets while achieving the same coverage. In their work, they compare the presented method with other conformal perdition methods, namely Adaptive Prediction Sets. Their approach has been carefully followed in our study, with necessary adaptations made to fit our specific context.

Moving forward, we will refer to the output of the model $\mathcal{M}$ as $\hat{f}(x) \in \mathbb{R}^L$, where L represents the number of label classes. And $\alpha \in [0, 1]$ is the acceptable error rate.

2.1.1 The Naive Method

In this method, the prediction set is constructed by including the platt scaled classes from highest to lowest probability until the total sum exceeds $1 - \alpha$.

Platt Scaling is a method used to calibrate the output of a classification model so they are better aligned with the true probability distribution of the outcomes. The fundamental idea of Platt Scaling is to fit a logistic regression model to the outputs of the original model. The original model's outputs are used as the inputs to the logistic regression model, and true labels are the target variables. This essentially transforms the raw model outputs into probabilities distributed between 0 and 1. This method is used as a baseline method.

2.1.2 Adaptive Prediction Sets (APS)

This method was first introduced in [8] and aims to address the limitations of The Naive Method. In this method, the produced prediction sets are adaptive to the inputs, producing small sets for easy inputs and larger sets for more complex inputs.

To achieve this property, we perform a calibration step to obtain the appropriate threshold by using a separate calibration set $\{(x_i, y_i)\}_{i=1}^{N}$. We define the score function $s(x, y)$:

$$s(x, y) = \sum_{j=1}^{l} \hat{f}(x)_{\pi_j(x)}, \text{ where } y = \pi_l(x) \tag{2.2}$$

here, y is the target class and $\pi(x)$ is the permutation of $\{1, \ldots, L\}$ that sorts the class probabilities in descending order. In other words, we sum all class probabilities in descending order until we reach the true class. The score function gives us a notion of how the model is uncertain. Larger values of $s(x, y)$ mean worse model predictions.

We calibrate the model by computing the scores for each datapoint in the calibration set, then, we obtain the appropriate threshold $\hat{q}$ by computing the adjusted $1 - \alpha' th$ quantile of the scores:

$$\hat{q} = Quantile(s_1, \ldots, s_N; \frac{\lceil (N+1)(1-\alpha) \rceil}{N}) \tag{2.3}$$

The prediction set is the set of classes with scores less or equal to $\hat{q}$:

$$C = \{y : s(x, y) \leq \hat{q}\} \tag{2.4}$$

2.1.3 Regularized Adaptive Prediction Sets (RAPS)

In [1], the authors introduced a subtle method to produce the smallest possible prediction sets. This is done by adding a regularization term to the score function in equation 2.2. In more details, consider $o_x(y)$ as the rank of the true target value y when sorted according to the class probabilities and $p_x(y)$ as the probability of the target class y, then the new score function is defined as:

$$s(x, y) = (\sum_{j=1}^{l-1} \hat{f}(x)_{\pi_j(x)}) + p_x(y) \cdot u + \lambda \cdot (o_x(y) - k_{reg})^+ \tag{2.5}$$

where u is a discrete random variable $u \in [0, 1]$. If $u = 1$, the target class probability will be added to the first term in equation 2.5, making it identical to equation 2.2. This is used to achieve a coverage of exactly $1 - \alpha$, as sets with size $l - 1$ tend to undercover and sets with size l tend to overcover. The regularization hyperparameters λ and k_{reg} are selected adaptively through an additional data splitting step. To elaborate briefly, using a small holdout set $\{(x_i, y_i)\}_{i=1}^{m}$, k_{reg} is the $\lceil (1 - \alpha)(1 + m) \rceil$ largest $o_x(y)$ and λ is picked using coarse grid search for the value that achieves smallest set size.

2.2 Monte Carlo Dropout

Monte Carlo (MC) Dropout is a subtle method introduced in [4] for measuring neural network uncertainty. The method leverages dropout layers to simulate ensemble learning and estimate uncertainty.

Dropout is a regularization technique used during training to prevent overfitting. It randomly drops a fraction of neurons in a given layer during each forward pass, encouraging the model to learn a more robust and generalizable representation. Usually, dropout is turned off during evaluation or inference to utilize the full parameters of the model. In Monte Carlo Dropout, dropout is also applied during inference in each forward pass. Instead of making a single prediction with all neurons, the model makes multiple predictions with different neurons deactivated each time. These multiple predictions can then be thought of as coming from an ensemble of "thinned" versions of the original model, providing a measure of variability or uncertainty in the predictions. While traditional ensemble methods require training multiple models on different subsets of the training data, Monte Carlo Dropout has an advantage by being more computationally efficient than the ensemble approach.

In the case of multi-class classification, the mean of each class probability is used for inference. More formally, Assume that we perform B forward passes through the model (with dropout enabled) and generate B predictions for a single datapoint, and let $\{(x_i, y_i)\}_{i=1}^{N}$ be a test set where $y \in \mathbb{R}_i^L$ and L representing the number of classes. The mean of class probabilities can be obtained as follows:

$$\bar{\hat{f}}(x_i) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}(x_i)_b \tag{2.6}$$

The variance σ^2 of each class probability across B samples can be used to provide an uncertainty estimate, where large variance encodes large uncertainty:

$$\hat{\sigma}^2(x_i) = \frac{1}{B-1}\sum_{b=1}^{B}(\hat{f}(x_i)_b - \bar{\hat{f}}(x_i))^2 \tag{2.7}$$

A $(1-\alpha)$ % prediction set can be constructed, similar to earlier, by including the classes from highest to lowest mean probability until the total sum exceeds $1-\alpha$. Throughout the implementation, we observed that this method tends to overcover by consistently producing sets with coverage greater than $(1-\alpha)$. Therefore, we introduce a randomization step that randomly removes the last class added to the prediction set to avoid overcoverage. Algorithm 1 describes the procedure formally.

Algorithm 1 Prediction set using Monte Carlo Dropout

Require: *sorted_classes*: classes sorted in descending order.

```
rand ~ UnifInt(0,1)
prediction_set ← {}
total_probabilities ← 0
for class in sorted_classes do
    prediction_set.add(class)
    total_probabilities ← total_probabilities + class
    if total_probabilities ≥ 1 − α then
        if rand == 1 and size(prediction_set) > 1 then
            prediction_set.pop()                    ▷ Randomization step
        end if
        break
    end if
end for
```

2.3 Experiments and Evaluation

We now turn to evaluating the abovementioned methods by conducting a series of experiments. Using a test set $\{(x_i, y_i)\}_{i=1}^{10000}$, we run 50 trials and compute the median-of-means for coverage and prediction set size and F1. For conformal prediction, in each trial, we randomly sample two subsets of the test set: the first set of size 1000 for calibration and the other of size 9000 for testing. For Monte Carlo Dropout, we use the entire test set for evaluation and set the number of forward passes to $B = 50$.

We evaluate each method based on coverage, prediction set size, and the F1 score

	Accuracy	Coverage			Size		
Model	F1	Naive	APS	RAPS	Naive	APS	RAPS
`Cifar10ConvModel`	0.7	0.875	0.904	0.897	2.28	2.54	2.29

Table 2.1: Results on CIFAR10 test set. We report the achieved coverage and the average set size for the model using **conformal prediction** for $\alpha = 0.1$.

Model	Coverage	Size	F1
`Cifar10ConvModel`	0.884	2.12	0.698

Table 2.2: Results on CIFAR10 test set. We report the achieved coverage and the average set size for the model using **Monte Carlo Dropout** for $\alpha = 0.1$.

the the top class in the prediction set. Coverage is the proportion of predictions for which the prediction set covers the true class. Higher coverage is better. Prediction set size refers to the number of classes in the prediction set. A smaller prediction set is generally preferable (assuming coverage is adequate), making the prediction more specific. These two metrics combined keep us informed on the model behavior. After all, the model can achieve perfect coverage by consistently producing large sets.

Table 2.1 shows the results of using the different conformal prediction method and table 2.1 shows results for using Monte Carlo Dropout. We observe that conformal prediction yields slightly better coverage, while Monte Carlo Dropout might provide more precise predictions (smaller prediction set size). However, both methods yield similar F1 Scores, suggesting that they perform comparably regarding overall classification accuracy.

Additionally, to evaluate the performance of the above methods more comprehensively, by augmenting the test set using AugMix augmentation. AugMix augmentation introduced in [6] uses a mixture of multiple transformations on each image, thus generating new data instances that have not been seen during training, thus simulating out-of-distribution data.

Tables 2.1 and 2.3 shows the results of the augmented test set. We observe that both methods have decreased in performance. At the same time, conformal prediction keeps achieving the desired coverage, while Monte Carlo Dropout falls short of achieving so.

	Accuracy	Coverage			Size		
Model	F1	Naive	APS	RAPS	Naive	APS	RAPS
`Cifar10ConvModel`	0.545	0.843	0.902	0.899	3.25	4.06	4.06

Table 2.3: Results on augmented CIFAR10 test set. We report the achieved coverage and the average set size for the model using **conformal prediction** for $\alpha = 0.1$.

Model	Coverage	Size	F1
`Cifar10ConvModel`	0.807	2.6	0.558

Table 2.4: Results on augmented CIFAR10 test set. We report the achieved coverage and the average set size for the model using **Monte Carlo Dropout** for $\alpha = 0.1$.

To summarize the results, from the original test set results, conformal prediction demonstrated similar coverage rates with a slightly larger prediction sets across all methods (APS and RAPS) compared to Monte Carlo Dropout. This indicates the conservative nature of this conformal prediction. Interestingly, the F1 scores, indicative of overall classification accuracy, were comparable for both methods.

Upon augmenting, the performance decreased for both methods - as one might expect when faced with more complex data. Coverage rates and F1 scores dropped while average prediction set sizes increased, reflecting the uncertainty. Furthermore, conformal prediction provided better coverage at the expense of set size. At the same time, Monte Carlo Dropout failed to achieve the desired coverage. This indicates that uncertainty estimates from conformal prediction are more robust.

Chapter 3

Uncertainty Quantification in Regression

This section will specifically focus on applying conformal prediction to regression problems. Regression tasks can bear significant uncertainties due to noisy data, model inadequacies, and unpredictable variations in the target variable. We will examine how conformal prediction can be utilized to build robust regression models capable of quantifying this uncertainty.

We will explain the underlying theory and property of conformal prediction in the context of regression. Following this, we will provide a detailed walk-through of how to implement conformal prediction in the context of regression and demonstrate and compare its performance with other classical methods through real-world examples.

In this study, we follow the procedure in [7] to build a conformal prediction framework. The method is called conformalized quantile regression (CQR) which we utilize quantile regression (QR) to construct prediction intervals and measure the accuracy of a given prediction.

3.1 Quantile Regression

In the context of regression, a linear regression model is used to find the best-fitting line that minimizes the sum of squared differences between the observed and predicted values, thus, estimating the conditional mean of the dependent variable given the independent variables. On the other hand, quantile regression is a regression technique that extends the concept of linear regression by estimating the conditional quantiles of the dependent variable. While linear regression focuses on modeling the mean, quantile regression provides information about different quantiles of the response variable distribution. This is particularly useful when the relationship between the variables is not symmetric or when we are interested in understanding different points in the distribution.

It's important to state that quantile regression builds on the assumption that datapoints in a dataset $\{(x_i, y_i)\}_{i=1}^n$ are exchangeable, therefor it's worth explaining the notion of exchangeability. A sequence of random variables $x_1, x_2 \ldots, x_n$ is said to be exchangeable if their joint probability distribution remains unchanged under any permutation of the indices. That is, for any permutation π of the indices 1, 2, ..., n, the joint distribution of $(x_1, x_2, \ldots, x_n)$ is the same as the joint distribution of $(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$. In practical terms, when data points are exchangeable, datapoints can be shuffled without changing the statistical properties of the data set. For example, time series datasets are often not exchangeable because of the temporal dependencies and sequential nature of time series data.

We will show in more detail how quantile regression works compared to the archetypal linear regression model. Suppose we have the following dataset $\{(x_i, y_i)\}_{i=1}^n$ with inputs $x \in \mathbb{R}^{d \times n}$ and targets $y \in \mathbb{R}^n$ and we want to come up with a model of the inputs x and tunable parameters $\phi \in \mathbb{R}^d$ that best represent y. We want ϕ to explain the data as well as possible such that:

$$y_i = x_i \phi + \epsilon_i \tag{3.1}$$

where $\epsilon \in \mathbb{R}^n$ is a noise term.

In the case of linear regression, ϕ can be estimated using the Least Square method by minimizing the sum of squared residuals:

$$\hat{\phi} = \arg\min_{\phi} \sum_{i=1}^{n} (y_i - x_i \phi)^2 \tag{3.2}$$

In the case of quantile regression, we want to come up with a model that best

represents a given quantile τ, such that:

$$q_\tau(x) = x\phi + \epsilon \tag{3.3}$$

The parameter ϕ can be estimated using the *pinball loss* function:

$$\hat{\phi} = \arg\min_{\phi} \sum_{i=1}^{n} \rho_\tau(y_i, x_i\phi) \tag{3.4}$$

where

$$\rho_\tau(y, x_i\phi) = \begin{cases} \tau(y - x_i\phi) & \text{if } y - x_i\phi > 0, \\ (1-\tau)(y - x_i\phi) & \text{otherwise} \end{cases} \tag{3.5}$$

Equation 3.5 can be considered a generalization of the weighted L1 norm function that is modified to target a given quantile τ.

We will now give a procedure to construct a prediction interval using quantile regression. We start by defining upper and lower quantiles. For $(1-\alpha)\%$ prediction interval, we have the upper quantile $\tau_{up} = 1 - \frac{\alpha}{2}$ and the lower quantile: $\tau_{lo} = \frac{\alpha}{2}$. Next, for each quantile, we fit a quantile regression model and estimate $\hat{q}_{\tau_{up}}(x)$ and $\hat{q}_{\tau_{lo}}(x)$. The prediction interval for a new observation is the range between the predicted value from the lower quantile model and the predicted value from the upper quantile model. Thus, the $(1-\alpha)\%$ prediction interval becomes:

$$C(x) = [\hat{q}_{\tau_{lo}}(x), \hat{q}_{\tau_{up}}(x)] \tag{3.6}$$

It's worth noting that $\hat{q}_{\tau_{lo}}(x)$ and $\hat{q}_{\tau_{up}}(x)$ in equation 3.6 are only estimates of the true quantiles. Therefore, it cannot be guaranteed that y will fall within the predicted range $(1-\alpha)\%$ of the time.

3.2 Conformalized Quantile Regression

We now turn to conformalize the prediction (CQR) interval in section 3.1. Assuming we have pretrained models that estimate $\hat{q}_{\tau_{up}}(x)$ and $\hat{q}_{\tau_{lo}}(x)$, following the conformal prediction framework, we start be defining a score function $s(x, y)$, such that larger scores encode large distance between y and it's nearest quantile prediction interval bound.

$$s(x, y) = \max\{\hat{q}_{\tau_{lo}}(x) - y, y - \hat{q}_{\tau_{up}}(x)\} \tag{3.7}$$

Next, we calibrate the model by computing the score of each datapoint in a calibration set $\{(x_i, y_i)\}_{i=1}^{N}$ and compute the adjusted $(1-\alpha)'th$ quantile $\hat{q}$ as follows:

$$\hat{q} = Quantile(s_1, \ldots, s_C; \frac{\lceil (N+1)(1-\alpha) \rceil}{N}) \tag{3.8}$$

Finlay, the prediction interval for a given x becomes:

$$C(x) = [\hat{q}_{\tau_{lo}}(x) - \hat{q}, \hat{q}_{\tau_{hi}}(x) + \hat{q}] \tag{3.9}$$

The effect of conformalizing a quantile regression model is that the upper and lower prediction interval bounds shift by $\hat{q}$ to obtain the desired coverage. From equations 3.7-3.9 we observe the following:

- If $\hat{q}$ is positive, the interval in equation 3.6 has coverage less than $(1-\alpha)$ and the interval in equation 3.9 becomes wider.
- Conversely, if $\hat{q}$ is negative, the interval in equation 3.6 has coverage greater than $(1-\alpha)$ and the interval in equation 3.9 shrinks.

The prediction interval in equation 3.9 has the following property:

$$\mathbb{P}[y \in C(x)] \geq 1 - \alpha \tag{3.10}$$

In other words, $C(x)$ has $1-\alpha$ marginal coverage. For a detailed proof of the property in equation 3.10, the reader can refer to [7].

3.3 Method

As previously discussed, quantile regression extends the regression model estimate to conditional quantiles. Traditional quantile regression techniques, while powerful, can be limited in their ability to model complex, non-linear relationships. Neural networks, with their layered structure, can be a solid alternative. Therefore, we will build conformalized quantile regression models using neural networks and regression trees as base models. In the subsequent sections, we explain each model's architecture and training process.

3.3.1 Models

Qunantil Neural Net We Build the neural network using 1 hidden layer with 64 nodes. We extend the output layer to have two outputs corresponding to upper and lower quantiles.

The model is trained through 100 epochs, where the training set is divided into batches of 32 samples each. We use `Adam` optimizer to adjust the model weights to minimize quantile losses. learning rate is set to $5 \cdot 10^{-4}$ and weight decay with value of 10^{-6} is applied during each optimization step.

Gradient Boosting Qunatile Reggressor Gradient Boosting is a type of boosting algorithm that uses decision trees as its base learners. It works by sequentially training new decision trees to correct the mistakes made by previous trees to improve the model's overall predictive power. We use the implementation provided by the `scikit-learn` library [1] which uses the algorithm introduced in [3].

CatBoosting Qunatile Reggressor CatBoost is a widely used algorithm for gradient boosting on decision trees. Developed by the search engine Yandex [2], it is especially powerful in handling categorical variables.

For neural networks, we adapt the model to quantile regression by changing the loss function to a *pinball loss* function introduced in section 3.1, which calculates the difference between the predicted and the actual quantile values for each instance in the dataset. Additionally, we modify the output layer to have two outputs (one for each quantile).

For tree-based models, we train a model for each quantile $\tau \in \{0.05, 0.95\}$ to construct the prediction interval. We use random search cross-validation to find the optimal hyperparameters. In randomized search, sampling with replacement is used to select the best parameter from a hyperparameters space. Table 3.1 shows the hyperparameter space used to train each model.

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html`

[2] `https://yandex.com`

Parameter	Distribution
max leaf nodes	Uniform[10, 50]
max depth	UniformInt[3,20]
number of estimators	UniformInt[50,300]
learning rate	Uniform[0,1]

Table 3.1: Hyperparameter space used for tree-based model parameter tuning.

3.3.2 Datasets

In this study, we will use two datasets, the California Housing Prices[3] and the Red Wine Quality datasets[4], to benchmark and evaluate the performance of conformalized quantile regression.

California Housing Prices data pertains to the houses found in a given California district and some summary stats about them based on the 1990 census data. The target variable is the median house value for California districts, expressed in hundreds of thousands of dollars ($\$100,000$).

The Red Wine Quality dataset contains the physicochemical properties of red wines. Our target variable is the quality score, ranging from 0 (very bad) to 10 (very good).

3.4 Experiments and Evaluation

We will now proceed to the evaluation of conformalized quantile regression. In this section, we evaluate the performance by conducting various experiments to assess the performance rigorously. We evaluate the methods by measuring the coverage and prediction interval length. Those metrics ensure that prediction intervals are accurate and as precise as possible.

[3]Dataset obtained from `https://www.kaggle.com/datasets/camnugent/california-housing-prices`

[4]Dataset obtained from `https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009`

Model	Avg. length	Coverage	α
`QR GradientBoostingRegressor`	1.09	0.432	0.05
`QR GradientBoostingRegressor`	0.816	0.361	0.1
`QR GradientBoostingRegressor`	0.588	0.283	0.2
`QR CatBoostingRegressor`	0.697	0.282	0.05
`QR CatBoostingRegressor`	0.508	0.23	0.1
`QR CatBoostingRegressor`	0.489	0.224	0.2
`QR QuantileNet`	2.22	0.723	0.05
`QR QuantileNet`	1.63	0.625	0.1
`QR QuantileNet`	0.957	0.387	0.2

Table 3.2: Results on California Housing test set We report average prediction interval length and coverage for the **quantile regression** models.

3.4.1 Experiments

We evaluate the coverage property by experimenting with $\alpha \in \{0.05, 0.1, 0.95\}$. We run 100 trials in each experiment and compute the median-of-means for coverage and average interval length. In each trial, we randomly sample two subsets from the test set $\{(x_i, y_i)\}_{i=1}^{T}$ for calibration and testing.

Tables 3.2 3.3 shows average prediction interval length and coverage on the California Housing dataset. Similarly, tables 3.4 3.5 show average prediction interval length and coverage on the Wine Quality dataset.

The results show that for both datasets, the CQR predictors consistently achieve the desired coverage for each model across all α levels. We also see that QR predictors fall short of reaching the desired coverage. However, the length of prediction intervals from CQR is generally larger than those produced by QR, which indicates a trade-off between coverage and precision. Comparing across models, `CatBoostRegressor` generally gives the smallest length prediction intervals, especially when using QR. However, it doesn't always provide the desired coverage levels, particularly at lower α levels. This indicates that while its intervals might be more precise, they may not be as reliable.

Model	Avg. length	Coverage	α
CQR GradientBoostingRegressor	3.0	0.949	0.05
CQR GradientBoostingRegressor	2.58	0.9	0.1
CQR GradientBoostingRegressor	2.05	0.8	0.2
CQR CatBoostingRegressor	3.03	0.949	0.05
CQR CatBoostingRegressor	2.45	0.899	0.1
CQR CatBoostingRegressor	2.02	0.8	0.2
CQR QuantileNet	3.78	0.951	0.05
CQR QuantileNet	3.27	0.902	0.1
CQR QuantileNet	2.42	0.8	0.2

Table 3.3: Results on California Housing test set We report average prediction interval length and coverage for $\alpha \in \{0.05, 0.1, 0.2\}$ for the **conformalized quantile regression** models.

Model	Avg. length	Coverage	α
QR GradientBoostingRegressor	1.81	0.74	0.05
QR GradientBoostingRegressor	1.03	0.62	0.1
QR GradientBoostingRegressor	0.618	0.49	0.2
QR CatBoostingRegressor	0.495	0.397	0.05
QR CatBoostingRegressor	0.482	0.343	0.1
QR CatBoostingRegressor	0.738	0.42	0.2
QR QuantileNet	2.27	0.873	0.05
QR QuantileNet	1.75	0.757	0.1
QR QuantileNet	1.07	0.56	0.2

Table 3.4: Results on Red Wine Quality test set. We report average prediction interval length and coverage for the **quantile regression** models.

3.4.2 Case Study: Estimating Discharge Time of Cargo Units

We will apply the methods introduced on a real-world dataset obtained from the Danish shipping and logistics company DFDS [5]. The goal of the study is to predict the time needed to discharge (offload) a cargo unit from a vessel given certain features of the container and the vessel transporting cargo.

In this study, we focus on implementing Feature-tokenizer Transformer (FT-Transformer) model as it's shown promising results when applied to tabular data. In addition, we compare it's performance to the models in section 3.3.

[5] https://www.dfds.com/en

Model	Avg. length	Coverage	α
CQR GradientBoostingRegressor	4.03	0.953	0.05
CQR GradientBoostingRegressor	2.05	0.91	0.1
CQR GradientBoostingRegressor	1.54	0.803	0.2
CQR CatBoostingRegressor	2.72	0.955	0.05
CQR CatBoostingRegressor	1.9	0.907	0.1
CQR CatBoostingRegressor	1.41	0.793	0.2
CQR QuantileNet	3.22	0.96	0.05
CQR QuantileNet	2.6	0.918	0.1
CQR QuantileNet	2.03	0.813	0.2

Table 3.5: Results on Red Wine Quality test set. We report average prediction interval length and coverage for $\alpha \in \{0.05, 0.1, 0.2\}$ for the **conformalized quantile regression** models.

FT-Transformer, first presented in [5], is a revised version of the original Transformer model, outlined by [11], which is optimized for tabular data handling. It converts all input attributes (numerical and categorical) into a unified embedding. This embedding is then run through a Transformer layer to attain contextualized embeddings. The Classifier Token's (CLS) final depiction is funneled into a linear layer, where the end prediction is generated. One significant advantage of using an FT-Transformer is that the numerical features are passed to the transformer layers together with categorical features. Thus, more context is gained.

The tuning of the model hyperparameters is based on their performance on the validation set. As Transformer models are typically resource-intensive, our tuning strategy is guided by time and hardware constraints, leading us to opt for a manual search approach to find the optimal hyperparameters.

Training model, spanning 65 epochs, takes approximately 5.5 hours on an Apple Pro equipped with an M1 chip and 16GB of memory. The model utilizes a multi-head self-attention mechanism with 8 heads. The Rectified Linear Unit (`ReLU`) function is the activation function, and Kaiming initialization is employed to set the weights. The optimizer used was `Adam`, and the Mean Squared Error was employed as the loss function. The learning rate was set at 10^{-4}, and the batch size was 64. Due to the considerable time required for training, we didn't explore training beyond 60 epochs. Nonetheless, beyond the 50th epoch, we didn't observe any significant improvements in the results. The model has an overall 629313 trainable parameters.

Additionally, an FNN model with 5 hidden layers has been implemented to be compared with FT-Transformer and to assess whether FT-Transformer is too

complex for the given problem. We used the `ReLU` activation function with a dropout rate of 0.3. We used the `Adam` algorithm for optimization, and the Mean Squared Error was used as a loss function. We set the learning rate to 10^{-4} and the batch size to 64.

It was also desired to compare the above model to a tree-based model, as they are known to perform very well on tabular data. Therefore, we utilized the Gradient Boosting model. The model is trained according to the training process for tree-based models in section 3.3.

We now turn to evaluate the models. We utilize the R^2 metric to compare the performance of the regression models. Table 3.6 shows the DFDS test dataset results for the base models and their corresponding R^2 accuracy. We observe that FT-Transformer accounts for approximately 46.9% of the variation in the target variable and slightly outperforms the Gradient Boosting model, which is typically considered a good model for dealing with tabular data. FT-Transformer also outperforms the benchmark FNN model, which indicates that the model is not over complicated for the dataset. While R^2 scores are not close to 1, it's worth noting that in complex real-world problems, achieving a very high R^2 score can be challenging due to numerous factors, including noise in the data and non-linear relationships.

3.4.2.1 Analysing Models Uncertainty

Relying on the R^2 metric alone doesn't provide a comprehensive understanding of the model performance. Therefore, we'll look into the prediction more thoroughly and quantify the associated uncertainty. Figure 3.1 shows the prediction distribution for FT-Transformer on the DFDS test dataset. We observe that the distribution is centered at 0 with mean $\hat{\mu} = 0.41$ and variation of $\hat{\sigma} = 58.86$. The High prediction variability indicates high model uncertainty for some predictions.

We utilize the approach detailed in section 3.3 to quantify the model uncertainty, and similarly, we adapt the base models to quantile regression and conformalize them in a manner consistent with the method introduced.

Based on the results presented in table 3.7, it's clear that both the QR models fail to meet the desired coverage. while CQR models achieve the desired coverage by widening the prediction intervals.

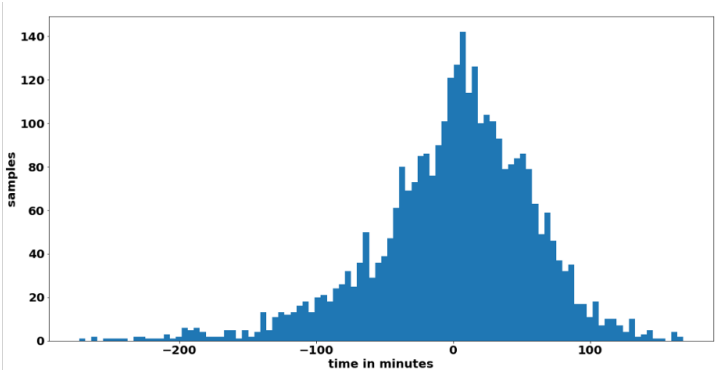


Figure 3.1: Prediction distribution for FT-Transformer on DFDS test dataset.

Model	R^2
`FT-Transformer`	0.469
`FNN`	0.418
`Gradient Boosting`	0.402

Table 3.6: Results on DFDS test set. We report the R^2 accuracy for each of the base models.

Model	Avg. length	Coverage
`QR FT-Transformer`	154.375	0.584
`QR Gradient Boosting`	85.896	0.313
`CQR FT-Transformer`	218.776	0.804
`CQR Gradient Boosting`	263.511	0.795

Table 3.7: Results on DFDS test set. We report average prediction interval length and coverage for $\alpha = 0.2$ for the **quantile regression** and **conformalized quantile regression** models.

Chapter 4

Uncertainty Quantification in Time Series

In the realm of time series analysis, predicting future values accurately is a challenging task due to the inherent complexity and temporal dependencies present in the data. Traditional approaches often rely on point predictions, estimating the expected value at a given time. However, these methods fail to capture the uncertainty associated with the predictions, which is crucial in decision-making processes and risk assessment.

To address this, we will explore conformal prediction to establish confidence interval quantifying uncertainty in time series prediction. By incorporating the temporal dependencies and sequential nature of time series data, conformal prediction provides reliable and calibrated measures of uncertainty.

This section introduces the principles and methodologies of conformal prediction in the context of time series analysis. We explain the underlying theory and techniques employed in constructing prediction intervals that account for uncertainty in time series prediction. Lastly, we implement the proposed method and evaluate thoroughly.

4.1 Conformal Prediction in Time Series

When estimating uncertainty in regression models, quantile regression has emerged as a popular approach that can provide prediction intervals for different quantiles of the target variable. However, as discussed in section 3.1, quantile regression builds on the assumption that datapoints in a dataset are exchangeable, which is rarely the case when dealing with time series datasets. Therefore, we explore alternative methods that offer a different perspective on quantifying uncertainty. This section uses the estimated standard deviation σ using Monte Carlo Dropout for each a model prediction $\hat{f}$ as an uncertainty measure.

In more details, assume an uncertainty function $u(x)$ such that it's large when a model prediction $\hat{f}$ has large uncertainty and small otherwise. with this notion of uncertainty we define a score function $s(x, y)$ as follows:

$$s(x, y) = \frac{|y - \hat{f}(x)|}{u(x)} \tag{4.1}$$

where $\hat{f}(x)$ model's point prediction and y is the target value. Given a holdout calibration dataset of size $\{(x_i, y_i)\}_{i=1}^{n}$, we compute the score $s(x_i, y_i)$ associated with each datapoint. Next, following the conformal framework, we compute the adjusted $(1 - \alpha)'th$ quantile:

$$\hat{q} = Quantile(s_1, \ldots, s_n; \frac{\lceil (n+1)(1-\alpha) \rceil}{n}) \tag{4.2}$$

The prediction interval for a datapoint x_i become:

$$C(x_i) = [\hat{f}(x_i) - u(x_i)\hat{q}, \hat{f}(x_i) + u(x_i)\hat{q}] \tag{4.3}$$

We shall now briefly explain the validity of the prediction interval in (4.3). From equation (4.2) we get:

$$\mathbb{P}[s(x, y) \leq \hat{q}] \geq 1 - \alpha \tag{4.4}$$

By the definition of the score function $s(x, y)$ in (4.1):

$$\mathbb{P}\left[\frac{|y - \hat{f}(x)|}{u(x)} \leq \hat{q}\right] \geq 1 - \alpha \tag{4.5}$$

Which leads to:

$$\mathbb{P}[|y - \hat{f}(x)| \leq u(x)\hat{q}] \geq 1 - \alpha \tag{4.6}$$

The left-hand side of the equation represents the probability that the absolute difference between the true value y and the predicted value $\hat{f}(x)$ (residuals),

scaled by the uncertainty function $u(x)$, falls within the computed threshold $\hat{q}$. In other words, it quantifies the likelihood that the prediction error is within a specific level of uncertainty. The right-hand side of the equation specifies that this probability should be greater than or equal to $1 - \alpha$, where $1 - \alpha$ represents the desired coverage probability or confidence level. This ensures that the estimated prediction intervals have a high likelihood of containing the true value within the specified confidence level.

4.2 Method

This section presents the methodology and steps involved in implementing conformal prediction for time series.

4.2.1 Model

In the implementation, we use a LSTM model as base model. LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) architecture that has proven highly effective in modeling and analyzing time series data. It addresses one of the key challenges in traditional RNNs, namely the vanishing gradient problem, by introducing specialized memory cells that can retain information over long periods, making them well-suited time series analysis.

4.2.2 Uncertainty Function

There are many ways to construct the introduced uncertainty function $u(x)$ in section 4.1, such as: fitting a model to predict the residuals $|\hat{f}(x) - y|$ [9], measuring the variance of $\hat{f}(x)$ across an ensemble of models or measuring the variance of $\hat{f}(x)$ to small input perturbations. In this study, we construct $u(x)$ by measuring the standard deviation of $\hat{f}(x)$ using Monte Carlo Dropout.

In more details, for the each datapoint x_i we compute the standard deviation of B model predictions $\hat{f}(x)$ while randomly dropping out a fraction of nodes for each prediction. Formally:

$$\hat{\sigma}(x) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\hat{f}(x)_b - \bar{\hat{f}}(x))^2} \tag{4.7}$$

where $\bar{\hat{f}}(x)$ is the sample mean,

$$\bar{\hat{f}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}(x)_b \tag{4.8}$$

Algorithm 2 gives a formal description of the procedure.

Algorithm 2 Uncertainty function algorithm

Require: B: sample size, x: input, $\mathcal{M}$: estimator, d: dropout fraction $\in [0, 1]$
1: $\mathcal{M}.init(dropout\_fraction : d)$
2: $outputs \leftarrow \{\}$
3: **for** $b \leftarrow 1$ to B **do**
4: $\quad \hat{f}(x) \leftarrow \mathcal{M}.predict(x)$
5: $\quad outputs.add(\hat{f}(x))$
6: **end for**
7: $u(x) \leftarrow \hat{\sigma}(outputs)$

In this study, we choose the sample size $B = 10$ and the dropout fraction $d = 0.3$. As mentioned earlier, the stochastic behavior of randomly dropping out a fraction of nodes introdhttps://da.overleaf.com/project/6476f65f5700015d53e4fd96uces uncertainty in the predictions, and measuring the standard deviation helps quantify the extent of this uncertainty.

4.2.3 Time Series Dataset

We use Amazon Stock Price dataset[1] to test the proposed method on. The dataset describe the daily stock history of Amazon in the period between 1997-05-16 and 2022-09-19. The dataset contains 6378 rows and contains the following columns:

- `Date`: Date of Listing (YYYY-MM-DD)
- `Close`: close value of Amazon stock
- `Open`: open value of Amazon stock
- `High`: highest recorded price for the day
- `Low`: lowest recorded price for the day

[1]Dataset obtained from `https://www.kaggle.com/datasets/whenamancodes/amazon-stock-market-analysis-founding-years`

- `Adj Close`: modified closing price based on corporate actions
- `Volume`: amount of stocks sold in a day

For our experiment, we only use `Date` and `Close` value to predict the close value.

We prepare the dataset for LSTM model by creating inputs shuch that they include the closing value for the previous 7 days. This way, the LSTM model will gain more context and will better predict the closing value by learning the values of previous days. Formally, we denote the dataset by $\{(x_i, y_i)\}_{i=1}^{6378}$, where x_i is a vector in the input feature space, such as $x_i \in \mathbb{R}^7$, and $y_i \in \mathbb{R}$ is the target value associated with the input vector x_i.

Finally, we transform features by scaling them. This is done by translating each feature individually so that it is in the given range on the dataset in the $[-1, 1]$ range.

4.3 Experiments and Evaluation

In this section, we start by showing the evaluation result on a test set, and then we turn to an experiment to assess the performance rigorously.

In the following, we follow the method introduced in 4.2 using the LSTM model and Amazon Stock Price dataset from the section. The evaluation will be based on coverage and prediction interval length.

4.3.1 Test Set Evaluation

We evaluate the method on a hold-out test set of size 974, and we use a calibration set of size 300. Figure **??** shows each test datapoint's actual close, predicted close and conformal interval. The conformal model has a coverage of 0.894 and an average interval length of 0.014. From figure **??**, we observe that the interval length does not vary much across the dataset, which is not optimal, as we want to infer model uncertainty from the interval length. This behavior could be due to the uncertainty function $u(x)$ not reflecting the model uncertainty very well.

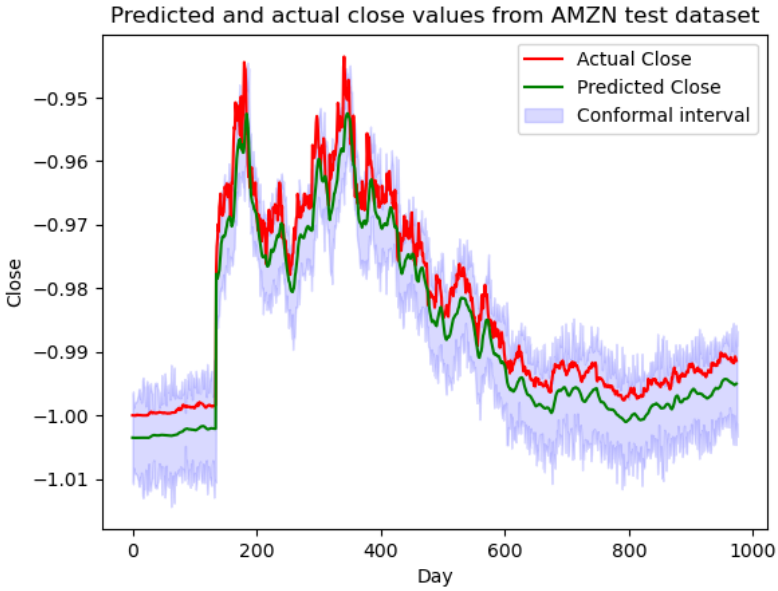


Figure 4.1: Result on Amazon Stock Price test set. We report the actual close, predicted close and conformal interval for each prediction.

Model	Avg. length	Coverage	α
LSTM_AMZN	0.011	0.807	0.2
LSTM_AMZN	0.014	0.9	0.1
LSTM_AMZN	0.018	0.953	0.05

Table 4.1: Results on Amazon Stock Price test set. We report average prediction interval length and coverage for $\alpha \in \{0.2, 0.1, 0.05\}$ for **conformalized LSTM model.**

4.3.2 Experiment

We now turn to testing the method more rigorously. We test the method with different α values to do so. We conduct an experiment and calculate the coverage and average prediction length for each $\alpha \in \{0.2, 0.1, 0.05\}$. We run 15 trials for each experiment and compute the median-of-means for coverage and interval length. In each trail, we randomly sample two subsets of the test set in: the first subset of size 225 is used for calibration, and the second subset of size 1049 is used for testing. Table 4.1 shows the results for each experiment. We observe that the method does deliver the desired coverage of $1 - \alpha$. Furthermore, we see that the average interval length does scale slightly with α.

It's worth noting that other types of uncertainty functions can be used to extend the capabilities of conformal prediction to better capture the uncertainties. Constructing the appropriate uncertainty function is the focus of many research areas. In [2], the authors propose assigning weights to conformal scores leveraging the recent trends in the data distribution. The proposed method can be used to handle distribution drifts (slowly varying changes in the data distribution) and is well suited for time series problems.

Chapter 5

Active Learning Using Uncertainty Estimates

In this section, we explore using uncertainty estimates in active learning. Active learning aims to solve one of the biggest challenges in machine learning, namely acquiring training data, especially when we deal with data-hungry machine learning models such as deep neural networks. In an active learning setting, the model -or learner- is trained on a small dataset and interactively asks a teacher or an expert to label selective data points. This gives the model some control over the learning process and what data to learn from. There are three different scenarios for implementing active learning:

- **Membership Query Synthesis** the learner synthesizes form input space, and then asks the teacher to label the synthesizes data. This has shown effectiveness in the case of a small dataset[13].
- **Pool-Based Sampling** the learner rank the entire unlabeled data by assigning a confidence score to each data point. This way, the learner can rank and query the data that it's least confidence about.
- **Stream-Based Selective Sampling** the learner evaluate the data points sequentially and independently asks the teacher for labels based on the information gained.

The performance and efficiency of an active learning setting also depend on the choice of data sampling method, i.e., what data to choose for labeling. Many methods have been proposed and studied in the active learning literature. For our purposes, we employ an uncertainty sampling method, where the learner queries the data that it's most uncertain about.

There are many strategies for uncertainty sampling that can be used in active learning to select the most informative samples for labeling. Most importantly, least confident sampling, sampling with the smallest best probability. Secondly, entropy sampling, the entropy formula is used on the output of each sample, and the sample with the highest entropy is considered to be the least certain.

In this study, we are interested in implementing and benchmarking the efficacy of using uncertainty measures from the conformal prediction in a pool-based sampling scenario. Our hypothesis is that the model poorly understands datapoints associated with large prediction sets. Thus, sampling by the largest prediction set increases the training efficiency and yields faster accuracy convergence.

5.1 Sampling Using Uncertainty Measures

In this section, we describe our active learning implementation and the proposed sampling by the largest prediction set strategy.

In the following, we denote the model by $\mathcal{M}$ and the dataset by $S = \{(x_i, y_i)\}_{i=1}^{n}$. Furthermore, the labeled pool $L \subseteq S$, the unlabeled pool $U \subseteq S$, and the sampling function $s(\mathcal{M}, U)$.

The algorithm starts by initializing the model, where the base model trains on a small, randomly selected subset of training data. Next, informative datapoints are selected to train the model on. In this step, the sampling function $s(\mathcal{M}, U)$ is used to select the most informative datapoints from the unlabeled pool U. After datapoints sampling, the unlabeled pool U is updated to exclude the sampled datapoints. The sampled datapoints are then added to the labeled pool, and the model is trained on the entire labeled pool. The process (sampling and retraining) is repeated for N times. Algorithm 3 describes the process formally.

The sampling function $s(\mathcal{M}, U)$ may change depending on the used uncertainty sampling strategy. In the least-confident strategy, datapoints with the smallest best probability are sampled. For entropy sampling strategy, the entropy formula is used on the output of each sample, and datapoints with the highest entropy are sampled. For sampling by the largest prediction set, datapoints

Algorithm 3 Active learning algorithm

Require: N: number of training rounds, $\mathcal{M}$: estimator, L: labeled set, U: unlabeled set, s: query algorithm

1: **for** $i \leftarrow 1$ to N **do**
2: $Sample = s(\mathcal{M}, U)$
3: $U = U \cap Sample$
4: $L = L \cup Sample$
5: $\mathcal{M}.train(L)$
6: **end for**

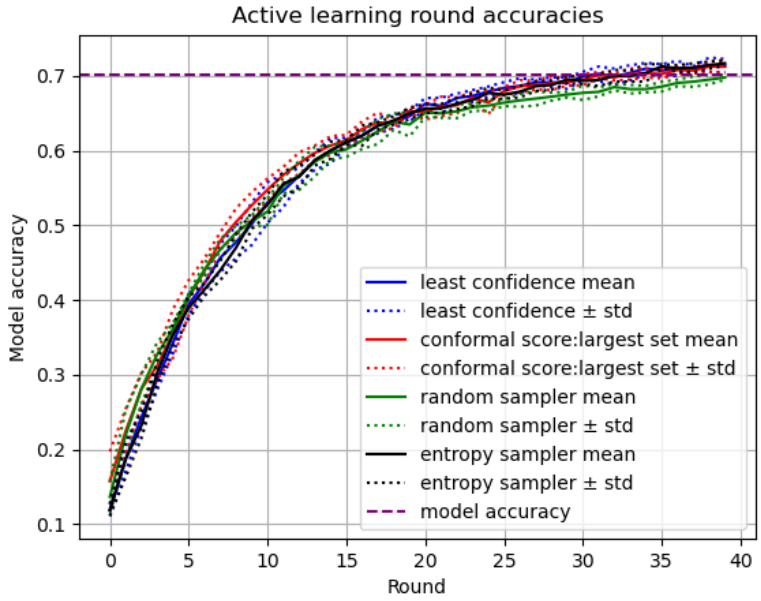
with the largest prediction sets are sampled.

5.2 Experiments and Evaluation

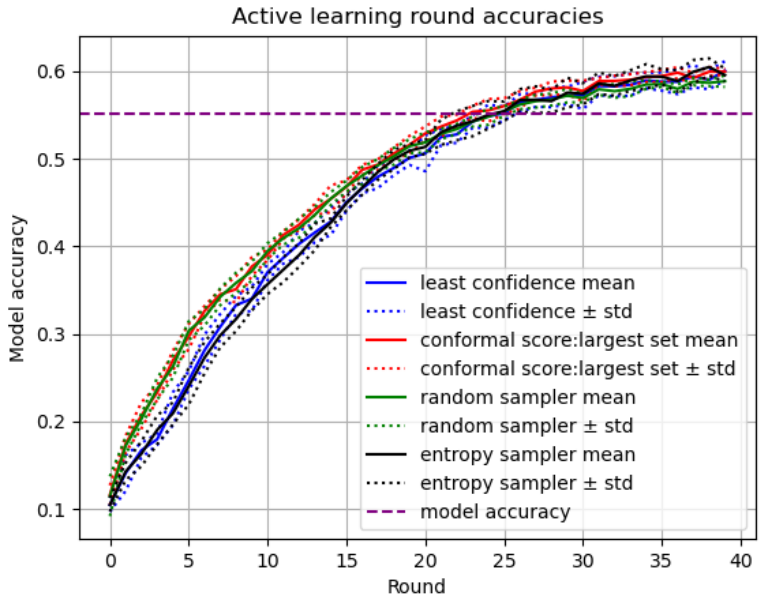
We now describe the experimental setup used to evaluate the abovementioned uncertainty sampling strategies. We test the algorithm on an image classification problem using the CIFAR10 dataset. We use the same model in section 2.3. The model is trained on a random subset of 1000 samples from the training set. Initially, the labeled pool L contains 0 samples while the unlabeled pool U contains 40000 samples. In each active learning round, we choose a sample size of 1000 samples. Furthermore, we use the test set from CIFAR10 to evaluate the performance after each active learning round. For active training, use `Adam` optimizer with a learning rate of 10^{-3}, and we train the model for 1 epoch in each active learning round.

The experiment is conducted using 5 trials and the mean and standard deviation accuracy for each uncertainty sampling strategy is reported. The performance of each strategy is evaluated based on how fast the model accuracy converges. As a baseline, we choose to compare against random sampling.

Figure 5.1 (a) shows the experiment results on the original test set. We observe a small but insignificant improvement in sampling by the largest prediction set over sampling by least-confidence. Upon augmentation, we observe from figure 5.1 (b) that the performance of sampling by least-confidence drops below the baseline while sampling by the largest prediction set stays consistent with the baseline. This indicates that sampling by the uncertainty estimates from conformal prediction is more robust and resilient towards distribution shifts. Furthermore, we observe from figure 5.1 that training on 60% of the training set using active learning with conformal prediction achieves the same accuracy as training on the entire training set.



(a) Results on original test set



(b) Results on augmented test set

Figure 5.1: Results on CIFAR10 test set with active learning. (a) original test set (b) augmented test set. The model is actively trained for 40 rounds using different uncertainty sampling.

Chapter 6

Conclusion

Throughout this study, we explored uncertainty quantification methods in various machine learning tasks. We focused on investigating the recently developed techniques in conformal predictions and their efficacy compared to similar methods in estimating model uncertainty.

In classification, we have implemented and analyzed several conformal prediction methods as well as Monte Carlo Dropout to produce prediction sets. We have shown that despite Monte Carlo Dropout's respectable performance, the prediction sets obtained through conformal prediction were consistently more robust across different test scenarios especially by using the Regularized Adaptive Prediction Sets method.

Shifting the focus to regression, we have explored how to adapt any base regression model to quantile regression and produce prediction intervals. Furthermore, we have shown how relying on quantile regression is not always reliable and better coverage is achieved by conformalizing the quantile regression model.

In time series analysis, we integrated Monte Carlo Dropout into the conformal prediction framework by using the estimated standard deviation of the predictions and showed its validity. However, we still need to explore the full extent of conformal prediction under distribution shift as it's an active research area [2] [10].

We have also shown that leveraging uncertainty estimates by conformal prediction can enhance the selection of informative instances during the active learning process. Furthermore, active learning guided by conformal prediction-based uncertainty estimates has consistently demonstrated robustness under varying inputs.

In conclusion, the applicability and effectiveness of conformal prediction were extensively explored throughout this study. We demonstrated its robustness and reliability compared to other methods. The intrinsic property of conformal prediction to provide valid prediction set under fairly general conditions was evident across all tasks, reinforcing its usefulness in uncertainty quantification.

Bibliography

[1] Anastasios Angelopoulos et al. *Uncertainty Sets for Image Classifiers using Conformal Prediction*. 2022. arXiv: 2009.14193 [cs.CV].

[2] Rina Foygel Barber et al. *Conformal prediction beyond exchangeability*. 2023. arXiv: 2202.13415 [stat.ME].

[3] Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine." In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451. URL: https://doi.org/10.1214/aos/1013203451.

[4] Yarin Gal and Zoubin Ghahramani. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. 2016. arXiv: 1506.02142 [stat.ML].

[5] Yury Gorishniy et al. *Revisiting Deep Learning Models for Tabular Data*. 2021. arXiv: 2106.11959 [cs.LG].

[6] Dan Hendrycks et al. *AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty*. 2020. arXiv: 1912.02781 [stat.ML].

[7] Yaniv Romano, Evan Patterson, and Emmanuel J. Candès. *Conformalized Quantile Regression*. 2019. arXiv: 1905.03222 [stat.ME].

[8] Yaniv Romano, Matteo Sesia, and Emmanuel J. Candès. *Classification with Valid and Adaptive Coverage*. 2020. arXiv: 2006.02544 [stat.ME].

[9] Steve Thorn. "Predicting uncertainty with neural networks". In: (2018).

[10] Ryan J. Tibshirani et al. *Conformal Prediction Under Covariate Shift*. 2020. arXiv: 1904.06019 [stat.ME].

[11] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[12] Alexander Gammerman Vladimir Vovk and Glenn Shafer. *Algorithmic Learning in a Random World*. 2005.

[13] Liantao Wang et al. "Active learning via query synthesis and nearest neighbour search". In: *Neurocomputing* 147 (2015). Advances in Self-Organizing Maps Subtitle of the special issue: Selected Papers from the Workshop on Self-Organizing Maps 2012 (WSOM 2012), pp. 426–434. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2014.06.042. URL: https://www.sciencedirect.com/science/article/pii/S0925231214008145.

Appendix

Technical Overview

This section provides an overview of some considerations that guided the code development process for this study.

We started this study with the ambition that users can regenerate the thesis's PDF from the data and code by running a single command. This was indented to support transparency and facilitate verifiable and auditable results. However, due to the time needed to run all the processes, we decided that each experiment could be verified and reproduced one at a time.

All neural network models used in this study were implemented using `PyTorch`. This open-source library was selected for its flexibility and compatibility with various machine learning tasks and models. All the uncertainty quantification methods were implemented from scratch without reliance on external libraries. This approach was chosen to be able to integrate different model architectures, where each model required different processes for training and prediction. Finally, we wanted others to be able to test and experiment with the methods easily. Therefore, no cloud setup is needed to run the code, and all processes (including model training and experiments), can be executed on a local machine.

The source code and usage instructions are open-source and available at `https://github.com/abdualrhman/uncertainty-quantification_for_deep_learning`.

Implementation Details

Conformal Methods

The class `ConformalClassifier` is built to conformalize a neural network classifier, taking a model, calibration dataloader and an error rate α as parameters and outputs prediction sets. Upon initialization, the class calibrates the model using the score function defined in equation 2.5 and computes $\hat{q}$. During the forward pass, the class constructs a prediction set from the base model outputs. A simple example is showen in listing 1.

```
import torch
import torchvision
from src.models.conformal_classifier import ConformalClassifier
from src.utils.utils import get_CIFAR10_img_transformer
# load datasets
calib_set = torchvision.datasets.CIFAR10(train=True, root='data/
    processed', download=True, transform=
    get_CIFAR10_img_transformer())
calib_loader = torch.utils.data.DataLoader(calib_set)
test_set = torchvision.datasets.CIFAR10(train=False, root='data/
    processed', download=True, transform=
    get_CIFAR10_img_transformer())
test_loader = torch.utils.data.DataLoader(test_set)
# conformalize
pretrained_model = Cifar10ConvModel()
conformal_model = ConformalClassifier(model=pretrained_model,
    calib_loader=calib_loader, alpha=0.1)
# prediction
for x, y in test_loader:
    prediction_set = conformal_model(x)
```

Listing 1: Conformalizing a neural net classifier

Similar to above, the class `CQR` is built to conformalize a quantile neural network regressor, taking a model, calibration dataloader and an error rate α as parameters and outputs prediction intervals. Upon initialization, the class calibrates the model using the score function defined in equation 3.7 and computes $\hat{q}$. During the forward pass, the class constructs a prediction interval from the base model outputs.

```
import torch
from src.models.CQR import CQR
from src.models.quantile_net import QuantileNet
from src.data.make_wine_quality_dataset import WineQuality
# load datasets
calib_set = WineQuality(train=True, in_folder='data/raw',
                        out_folder='data/processed')
calib_loader = torch.utils.data.DataLoader(calib_set)
```

```
test_set = WineQuality(train=False, in_folder='data/raw',
                       out_folder='data/processed')
test_loader = torch.utils.data.DataLoader(test_set)
# conformalize
pretrained_model = QuantileNet(input_size=11)
conformal_model = CQR(model=pretrained_model,
                      calib_loader=calib_loader, alpha=0.1)
# prediction
for x, y in test_loader:
    prediction_set = conformal_model(x)
```

Listing 2: Conformalizing a quantile neural net

The *Pinball loss* function in equation 3.5 is costume build and can be used by passing a list of desired quantiles as a parameter. See listing 3.

```
from src.models.quantile_net import QuantileLoss

loss_fn = QuantileLoss([0.05, 0.95])
```

Listing 3: Pinnball loss function

Active Learning

For active learning, the class `Oracle` can be used by passing the model, training set, test set, and sampling parameters. See listing 4.

```
import torchvision
from src.models.oracle import Oracle
from src.models.cifar10_conv_model import Cifar10ConvModel
from src.utils.utils import get_CIFAR10_img_transformer
# load datasets
train_set = torchvision.datasets.CIFAR10(
    train=True, root='data/processed', transform=
    get_CIFAR10_img_transformer())
test_set = torchvision.datasets.CIFAR10(
    train=False, root='data/processed', transform=
    get_CIFAR10_img_transformer())

model = Cifar10ConvModel()
oracle = Oracle(model=model, train_set=train_set, test_set=test_set
    , strategy='least-confidence', sample_size=1000)
# start active learning for 1 round
oracle.teach(1)
print(oracle.round_accuracies)
```

Listing 4: Training a model for 1 round using active learning

Dataset Descriptions

This appendix provides a brief overview of the three primary datasets used in our study.

CIFAR10

The CIFAR10 dataset (Canadian Institute For Advanced Research) is a well-known dataset used for image recognition tasks. It consists of 60,000 32x32 color images divided into ten different classes, with 6,000 images per class. The classes include objects such as airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is split into 50,000 training images and 10,000 test images. The relatively low resolution and balanced nature of the dataset make it a common choice for benchmarking image classification models.

California Housing

The California Housing dataset is a popular dataset for regression tasks. It contains information from the 1990 California census and includes features such as the median income, housing median age, average number of rooms per dwelling, average number of bedrooms per room, population, average occupancy, latitude, and longitude. The dataset's target variable is the median house value for California districts. This dataset is often used to create models that predict median house values based on other housing features.

Wine Quality Red

The Wine Quality Red dataset is used for both regression and classification tasks in machine learning. The dataset is related to red variants of the Portuguese "Vinho Verde" wine. It contains physicochemical data (11 features) such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol. The output variable based on sensory data is the quality of the wine on a scale from 0 (very bad) to 10 (very good). This dataset is commonly used to create models predicting wine quality based on physicochemical properties.

Attribute Name	Type	Description
disharge_time_minutes	Ratio	container discharge time (Target)
length	Ratio	Length of container
gross_weight	Ratio	Weight of container
parking_place	Nominal	Parking place of container offloaded
deck_on_vessel	Nominal	Deck on vessel container is offloaded from
place_on_board	Nominal	Position of the container on deck
deck_stowed_order	Nominal	Order by which container is offloaded by
is_reefer	Binary	Whether the container is refrigerated
is_hazardous	Binary	Whether the container is Hazardous
unitype_id	Nominal	The type of unit id in numbers

Table 1: DFDS dataset attributes.

DFDS: Cargo Discharge Time

The dataset obtained from DFDS includes 43,890 datapoints and each datapoint has 31 features. This dataset is tabular, having both categorical and numerical variables. 12 attributes from the dataset are used for modelling. The categorical attributes are transformed using both one-hot encoding and label encoding. See table 1 for a detailed description.