Confidence of deep learning model predictions

Author: Carol Guo

# **Project Overview**

## **The importance of confidence in deep learning**

In deep learning, uncertainty refers to the lack of knowledge or confidence in the model's predictions or parameters.

In recent years, confidence plays a crucial role in deep learning, as it provides valuable information about the reliability of model predictions, and there has been a growing interest in developing deep learning models that can estimate and propagate uncertainty, which can help improve model robustness and reliability in real-world applications.

## **Project Objectives**

This project is about implementing and developing three algorithms to assess the prediction confidence of deep learning models on two different datasets. By testing the algorithms’ performance on two datasets, the superiority of different methods can be verified.

The main objectives of this project include:

1. Load a deep learning model in ONNX format and convert it to a format compatible with a deep learning library (e.g., TensorFlow or PyTorch)
2. Develop different model-agnostic algorithms for assessing the confidence of the model's predictions
3. During model serving, apply the developed confidence assessment algorithms to the incoming batches of data and compute confidence scores for each prediction

The main idea is to explore and implement different techniques to estimate the confidence of the model's predictions, which can be helpful in various real-world applications, such as risk assessment, decision making, and anomaly detection.

## **Project Implementation**

Two functions are built for comparing the confidence of three algorithms:

### **Setup()**

The setup function is used to read the neural network model in the ONNX format, which is an open-source framework for exchanging deep learning models between different frameworks. After reading the .onnx file to obtain the model structure, the setup function accesses the graph structure through the model structure and then creates the underlying computational graph, finally returning the model structure and parameters for each layer. The output of the setup function should be available in the subsequent confidence assessment algorithms.

Details about the function process:

* Read the .onnx file to obtain the model structure
* Access the graph structure through the model structure
* Access all the nodes, inputs, and outputs of the entire network through the graph
* Through the node structure, you can obtain the parameter information of each OP
* In the graph structure, there are also initializer and value\_info defined, which respectively store the weight parameters of the model and the output information of each node

### **Confidence()**

The confidence function is different for different algorithms but the output is a list of confidence scores, one per sample. The results returned by this function are used for assessing the prediction confidence of deep learning models.

# **Algorithms**

## **Logit-based confidence**

### **Algorithm description**

Neural networks have achieved impressive accuracy in classification tasks, but they can still make costly mistakes. By knowing the uncertainty associated with the classification output of neural networks, we can identify when the model is more likely to make errors and improve the predictions further.

Several methods rely on softmax probabilities generated by neural networks to evaluate classification uncertainty. However, these probabilities are unreliable for some tasks such as medical diagnosis or fraud detection, where mistakes can be costly. To address this issue, an innovative method called logit uncertainty has been developed based on the logit outputs of neural networks.

Logit uncertainty employs a Gaussian Mixture model to capture the logit outputs of correctly predicted training samples for each class. The uncertainty values are modeled based on the probability density function of the Gaussian Mixtures. This approach helps classification models to detect when they are more likely to make mistakes and improve the accuracy of their predictions.

Logit values capture the degree of uncertainty in neural network classifications. This means that the logit outputs for a particular class should have similar logit values in each dimension of the logit vector. This logic follows from the idea that samples from the same class should share similar features. If the logit values of a sample differ significantly from the known logits of its predicted class, the prediction is considered to have low confidence.

To measure uncertainty for each class, Gaussian Mixture Models (GMMs) are fitted to the logit vectors of correctly classified samples. GMMs are used because they are universal approximations and, under certain assumptions, neural networks generate GMM-distributed outputs. One natural approach is to base the uncertainty measure on the density functions of the GMMs, meaning that the larger the density function value, the smaller the uncertainty. To deal with the extremely small values of density functions commonly observed in datasets, a score function is designed to make the values more manageable. Finally, the sigmoid function is applied to map the values obtained from the score function to the range between 0 and 1.

To interpret the result, knowing the uncertainty with the output, the higher the uncertainty, the more likely to make mistakes. Based on the logit value to get the uncertainty, the final uncertainty is between 0 and 1.

### **Model Implementation**

The neural network model built for implementing the algorithm is [here](https://raw.githubusercontent.com/YQ9818/bank-marketing/7f26fb0e24db7b49f2d01a3902a9d60f42b1a49e/Method/logit/my_model.onnx.svg).

This is a feedforward fully connected architecture. The model consists:

1. Input layer: dense\_66\_input
   * This is the input layer to the network. It has number of input features multiplied by 55, creating a 2D input tensor
2. Layer 1:
   * MatMul B <55\*64>: Matrix multiplication operation where the input is multiplied by a weight matrix of dimensions 55x64
   * add - B<64>: An addition operation, where a bias vector of length 64 is added to the result of the matrix multiplication
   * Relu: The activation function is applied element-wise to the output of the previous layer
3. Layer 2
   * MatMul B <64\*64>: Matrix multiplication operation with a weight matrix of dimensions 64x64.
   * add - B<64>: An addition operation, where a bias vector of length 64 is added to the result of the matrix multiplication
   * Relu: The ReLU activation function is applied again
4. Layer 3
   * MatMul B <64\*32>: Matrix multiplication operation with a weight matrix of dimensions 64x32
   * add - B<64>: An addition operation, where a bias vector of length 64 is added to the result of the matrix multiplication. (This appears to be incorrect. The bias vector should have the same length as the output of the previous layer, which should be 32.)
   * Relu: The ReLU activation function is applied
5. Output layer
   * MatMul B <32\*1>: Matrix multiplication operation with a weight matrix of dimensions 32x1
   * add B <1>: An addition operation, where a bias vector of length 1 is added to the result of the matrix multiplication
   * Sigmoid: The Sigmoid activation function is applied element-wise to the output of the previous layer
   * unk+7\*1: The output tensor has a single output value for each sample
   * dense\_69: The name of the output layer in the network

The model consists of an input layer, three fully connected hidden layers with ReLU activation functions, and a fully connected output layer with a Sigmoid activation function. The architecture is designed for a binary classification problem as the output layer has only one unit with a Sigmoid activation, which returns a value between 0 and 1, representing the probability of a positive class.

## **DKNN**

### **Algorithm description**

**DNN & KNN**

Deep Neural Network (DNN) is a type of neural network that is composed of many layers of interconnected nodes. The architecture of a DNN is designed to automatically learn representations of data by mapping input data to output predictions through a series of hidden layers. Each hidden layer uses a set of learned parameters to transform the input data into a more abstract and higher-level representation, which is passed to the next layer. The final layer produces the output prediction based on this processed representation of the input data.

DNN has gained increasing importance these days due to its ability to solve complex engineering problems and it enables innovative applications of machine learning like image recognition, machine translation, or malware detection. But despite the breakthroughs it enabled, the adoption of deep neural networks (DNNs) in security and safety-critical applications remains limited in part because they are often considered as black-box models whose performance is only partially understood and are controlled by a large set of parameters. Given the lack of robustness in adversarial settings and the general inability to rationalize its predictions, one paper exploits the structure of deep learning to enable new learning-based inference and decision strategies that achieve desirable properties such as robustness and interpretability.

**DKNN**

Deep k-Nearest Neighbors (DkNN) is a classification algorithm that combines the benefits of deep learning and k-NN (k-Nearest Neighbors) algorithms, which enforces conformity of the predictions made by a DNN on test inputs with respect to the model’s training data.

DKNNs use a deep neural network to learn the feature representations of the input data, but instead of making a direct prediction based on these representations, after training completes, the DKNN algorithm passes every training example through the model and saves each of the layer’s representations. This creates a new dataset, whose features are the representations and whose labels are the model predictions. Then for each layer in the DNN, the DkNN performs a nearest neighbor search to find training points for which the layer’s output is closest to the layer’s output on the test input of interest. The predictions are then made based on the labels of the nearest neighbors and can then analyze the label of these neighboring training points to ensure that the intermediate computations performed by each layer remain conformal with the final model’s prediction.

One of the benefits of a DKNN is that it can provide more interpretable results than a DNN, as the k-NN algorithm can provide explanations for why a particular prediction was made.

An additional benefit is the algorithm’s uncertainty metric, the conformity score. This score is the percentage of nearest neighbors belonging to the predicted class.

The integrity of the DkNN classifier is maintained when its prediction is supported by the underlying training data. This support is evaluated as level of “confidence” in the prediction’s agreement with the nearest neighbors found at each layer of the model and analyzed with conformal prediction.

Probabilities output by DNNs are commonly used as a proxy for their confidence. Yet, these probabilities are not faithful indicators of model confidence.

In the DkNN, the number of nearest neighboring training points whose label does not match the prediction made on a test input defines an estimate of the input’s nonconformity to the training data. The larger that number is, the weaker the training data support the prediction.

### **Model Implementation**

To compare the confidence output of different methods, the same neural network model built for implementing the algorithm is [here](https://raw.githubusercontent.com/YQ9818/bank-marketing/7f26fb0e24db7b49f2d01a3902a9d60f42b1a49e/Method/logit/my_model.onnx.svg) as the previous one for logit-based.

##### **Find K nearest neighbors based on LSH technique**

The DKNN algorithm can be improved by using LSH to speed up the nearest neighbor search in high-dimensional spaces. LSH can be used to efficiently find the k-nearest neighbors of a query point in the high-dimensional space learned by a deep neural network.

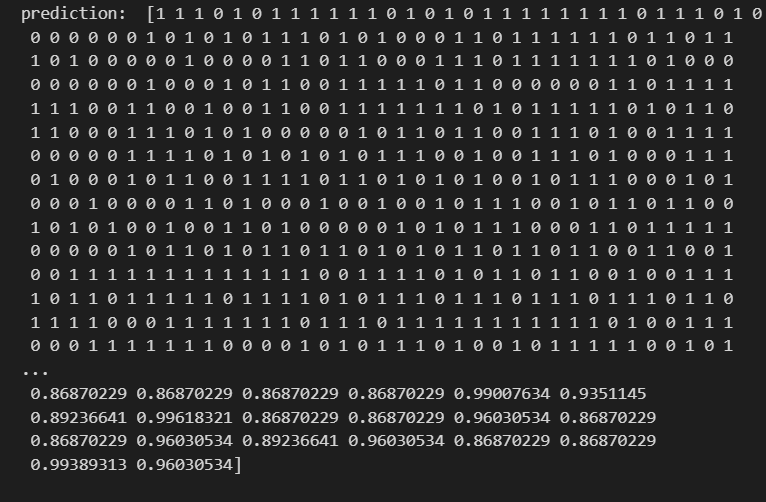
In DKNN with LSH, the high-dimensional space is divided into smaller subspaces using LSH functions, and each subspace is represented by a hash table. During training, the hash tables are populated with the training data and their labels. During testing, a query point is first transformed into the high-dimensional space using a trained neural network and then hashed using the LSH functions. The query point is then looked up in the hash tables corresponding to the subspace in which it falls. The k-nearest neighbors of the query point are found by comparing the query point to the points in the hash table, and the p-value is calculated as usual.

By using LSH to speed up the nearest neighbor search, DKNN with LSH can handle high-dimensional data more efficiently and accurately than traditional nearest neighbor methods. LSH reduces the search space and allows for fast retrieval of the k-nearest neighbors, which is particularly important in applications such as image and speech recognition, where the dimensionality of the data can be very high.

# **Comparison**

The logit-based method returns the confidence score as an uncertainty value, the higher the uncertainty, the more likely to make mistakes for a specific output class. The DKNN method returns the confidence score as the conformity score. This score is the percentage of nearest neighbors belonging to the predicted class. While the methods of measuring confidence are different between different algorithms, the output of confidence follows the same format.

Here is an example of the confidence output:



In this project, the logit-based method and DKNN output contain two arrays, the first array represents the predicted class (0 or 1), and the second array is the confidence score, respectively.

Given that the final result of the confidence function is a list of confidence scores, to compare the performance of different confidence assessment algorithms, some evaluation metrics that can aggregate the confidence scores into a single number needed to be used. This enables to us quantitatively compare the effectiveness of each method.

Some popular evaluation metrics that could be applied are:

1. **Accuracy**: We can calculate the accuracy of the model's predictions, considering only the predictions with confidence scores above a certain threshold. This will give us an idea of how well the algorithm can identify high-confidence predictions that are also correct
2. **Area Under the Receiver Operating Characteristic (ROC) Curve (AUC-ROC)**: This is a popular metric for classification problems. It plots the true positive rate against the false positive rate at different threshold settings. The AUC-ROC score can help us assess the trade-off between sensitivity and specificity for different confidence threshold levels
3. **Mean Average Precision (MAP)**: This metric takes into account both the precision and recall of the model's predictions. It is particularly useful when you have imbalanced classes, as it accounts for the false positives and false negatives in the predictions