Uncertainty Simple Perceptron for Noise and Outliers Prediction

Anonymous Author(s) Affiliation omitted for double-blind review

***Abstract*—Current machine learning models have tended to grow increasingly larger. Architectures such as Transformers rely on feature learning, often overlooking models based on feature engineering and mathematical modeling, which can achieve full precision using models without hidden layers. This work proposes a modification to the first machine learning model, the simple perceptron, by adding a mechanism to quantify its own uncertainty.**

This approach achieves 100% selective accuracy and 71.5% total accuracy with a computational cost of O(n), resulting in computational savings of over 99.99% compared to models like Transformer. The model was tested across multiple noisy environments, including Tesla, Apple, Microsoft, and Meta. This proposal establishes a new category of models, trained on an i5-12450 CPU with 32 GB of RAM in approximately three seconds.

***Index Terms*—Simple perceptron, Cover theorem, uncertainty, quantification, time complexity, time series, feature engineering, and noisy prediction,**

I. Introduction

Current models composed of multiple algebraic transformations (hidden layers) result in time complexity greater than O(n) for all predictions, which makes their training, inference, and fine-tuning costs excessively high.

These current models find patterns within a raw dataset by geometrically transforming the data and processing it layer by layer in search of patterns.

Feature learning can be replaced by feature engineering, where instead of using hidden layers to transform the data, mathematical formulas are used to represent these data (e.g., if you want to represent a point on a circle, you can use sine and cosine to represent coordinates).

In 1957, after the creation of the simple perceptron by Rosenblatt, the mathematician Cover (1965) proposed — "complex pattern classification problems, cast in a high-dimensional space nonlinearly...". This means that in high-dimensional projections, a simple model increases the likelihood of finding linear separability in a dataset.

II. Background

Linear models such as the simple perceptron (Rosenblatt, 1957) or linear regression (Legendre, 1805) share the property of making predictions within a binary window (0, 1).

These linear models operate under the "general linear combination formula" (Legendre, 1805). In the case of the simple perceptron, these linear combinations can be interpreted as the dot product of the weights of each feature and their corresponding input, plus the model's bias.

$$z = \sum_{i=1}^{n} w_i x_i + b$$

The output of this formula is passed through a function, with each model having its own activation (e.g., simple perceptron; step function).

$$\hat{y} = f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

III. Uncertainty in the Simple Perceptron

To quantify the uncertainty () in the simple perceptron, the geometry of the "step function" defines the point at which the weighted sum falls for positive (1) and negative (0) cases.

$$h(x) = \begin{cases} 1 & \text{if } x > \epsilon \\ 0 & \text{if } x < -\epsilon \\ 0.5 & \text{if } -\epsilon \leq x \leq \epsilon \end{cases}$$

The uncertainty () is a constant that its defined before each prediction (creating a grey area).

Thus, the output will be positive (1) if the continuous value (x) is greater than the uncertainty (); in the event that the output (x) is less than the negative uncertainty (), it will be a negative prediction (0). Finally, if neither of those two conditions are false, epsilon will have been fulfilled, meaning the prediction falls into the area of uncertainty ().

To compute the model's uncertainty, the standard deviation () was used across all the training epochs, multiplying the error rate (error) by a variable to control the model's sensitivity (), represented as:

$$\epsilon = \frac{\sigma}{\text{error} \cdot \Delta}$$

IV. Learning Rule for the Simple Perceptron

For each learning epoch, the model uses the Perceptron learning rule (Rosenblatt, 1958) to adjust its weights. This adjustment is calculated by multiplying the learning rate () by the difference between the true label (y) and the perceptron's output (ŷ), based on the binary step function's output (0 or 1).

$$w_i \leftarrow \eta(y - \hat{y})x_i$$

$$b \leftarrow b + \eta(y - \hat{y})$$

V. Model Vectorization Representation

To represent the vector algebraically, various transformations are applied to the data in order to construct the input vector. This vector should contain the greatest amount of diverse information possible, so as to increase the likelihood of achieving linear separability within our dataset.

- The technical indicators create a three-dimensional space that reflects momentum, the strength of price movements, and a time-series exponential component.

$$\%K = \frac{\text{close} - \text{low}_n}{\text{high}_n - \text{low}_n} \cdot 100$$

$$RS = \frac{\text{Avg. Gain}}{\text{Avg. Loss}}$$

$$RSI = 100 - \frac{100}{1 + RS}$$

$$EMA_{\text{initial}} = SMA = \frac{P_1 + P_2 + \cdots + P_n}{n}$$

$$EMA_t = EMA_{t-1} + \alpha(P_t - EMA_{t-1})$$

$$\alpha = \frac{2}{n+1}$$

- In the representation of cyclic time encoding, these models learn to develop a sense of time (e.g., month, week, hour, etc.).

$$v = \frac{2\pi \cdot r}{T}$$

$$\sin(v), \quad \cos(v)$$

- The vectorization uses the z-score normalization, where volume, transaction count, and the volume-weighted average price are normalized.

$$z = \frac{x - \mu}{\sigma}$$

VI. Training and Hardware

The model was trained and tested against TESLA's time series, using a data window starting from November 2022, up to November 2025. The data was divided from 2022, with 73% used for training and the remainder for subsequent testing.

I trained the model using an Intel i5-12450 and 32 GB of RAM. The training was stopped at 43 epochs due to early stopping, and the model was restored from epoch 34 with an error rate of 28.5%. The training process took a total of 3.46 seconds. Additionally, the vectorization of the data set, consisting of approximately 40,000 vectors, took about 6.3 seconds.

VII. Results

Across more than ten different noisy markets, the model's results on the test set were an inference time of approximately 0.0001 seconds per prediction with an accuracy rate of 100%, filtering out 82.43% of the noise.

VIII. Comparation

In this comparison (benchmark), the results of the uncertainty simple perceptron model are shown against the following models (metrics extracted and averaged from their official papers): TranAD (Tuli et al., 2022), LSTM Autoencoder (Hochreiter Schmidhuber, 1997), GRU (Cho et al., 2014), and Isolation Forest (Liu et al., 2008).

TABLE I
Percentage of incorrect predictions in the training phase.

Model	**Error Rate**
Uncertainty Simple Perceptron	0.2858
TranAD (Transformer)	∼0.12 - 0.15 (F1-Score)
LSTM-Autoencoder	∼0.21 - 0.25
GRU	∼0.22 - 0.26
Isolation Forest	∼0.32 - 0.38

Although the Uncertainty Simple Perceptron shows better metrics in the benchmark, it is important to

TABLE II
PERCENTAGE OF POSITIVE PREDICTIONS IN THE PREDICTED SET.

Model	**Accuracy**
Uncertainty Simple Perceptron	100%
TranAD (Transformer)	∼85–92%
LSTM-Autoencoder	∼78–84%
GRU	∼75–82%
Isolation Forest	∼65–72%

TABLE III
ESTIMATED TRAINING TIME ON THE TRAINING SET.

Model	**Training Time (seconds)**
Uncertainty Simple Perceptron	∼3.784
TranAD (Transformer)	∼784.89
LSTM-Autoencoder	∼267.89
GRU	∼78.84
Isolation Forest	∼5.84

understand that this model, thanks to its ability to estimate uncertainty over the data, is able to filter out a high rate of noise (approximately 87%) from the data it does not understand.

This allows the model to know when something is completely certain; however, it cannot predict everything, always and in all cases.

IX. CONCLUSION

In this work, the "Uncertainty Simple Perceptron" is presented, the first linear model with a computational cost of O(n).

This work validates Cover's theorem. This model is the first to achieve a 100% accuracy rate, obtained by filtering noise in one of the world's noisiest datasets (TESLA).

Finally, a cost savings greater than 99.999% was shown in inference, training, and fine-tuning costs compared to other alternatives.

REFERENCES

[1] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation", 2014.
[2] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition", 1965.
[3] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", 1997.
[4] A. M. Legendre, "Nouvelles méthodes pour la détermination des orbites des comètes. Courcier", 1805
[5] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation forest", 2008
[6] F. Rosenblatt, "The perceptron, a perceiving and recognizing automaton", 1957.
[7] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain", 1958.
[8] S. Tuli, G. Casale, and N. R. Jennings, "TranAD: Deep transformer networks for anomaly detection in multivariate time series", 2022.

TABLE IV
TIME COMPLEXITY OF EACH MODEL.

Model	**Time Complexity**
Uncertainty Simple Perceptron	$O(n)$
TranAD (Transformer)	$O(n^2)$
LSTM-Autoencoder	$O(n)$
GRU	$O(T \cdot (n^2 + n \cdot m))$
Isolation Forest	$O(i \cdot \psi \log \psi)$

GitHub repository omitted for double-blind review.