A review of Machine Learning methods for Transition edge sensor signal classification

September 15, 2023

1 Dataset

The data obtained from the Transition edge sensor (TES) takes the form of a 1D univariate time-series, $X(t) = [x_0, x_1, x_2...]$. The shape of the signal is a result of the electrical and thermal constant of the TES circuit [1]. At low repetition rates, such as 100kHz, this can be approximated by [2]:

$$C + 2A[exp(\frac{t - t_0}{\tau_{rise}}) + exp(\frac{t_0 - t}{\tau_{fall}})]^{-1}, \tag{1}$$

and at higher repetition rates (≥200kHz), where the previous tail overlaps due to the next signal arriving while the TES is still cooling, an extra term can be included to characterise this.

$$C + 2A_0[exp(\frac{t - t_0}{\tau_{rise}}) + exp(\frac{t_0 - t}{\tau_{fall}})]^{-1} + A_{-1}exp(\frac{t - t_{-1}}{\tau_{fall-1}}) \tag{2}$$

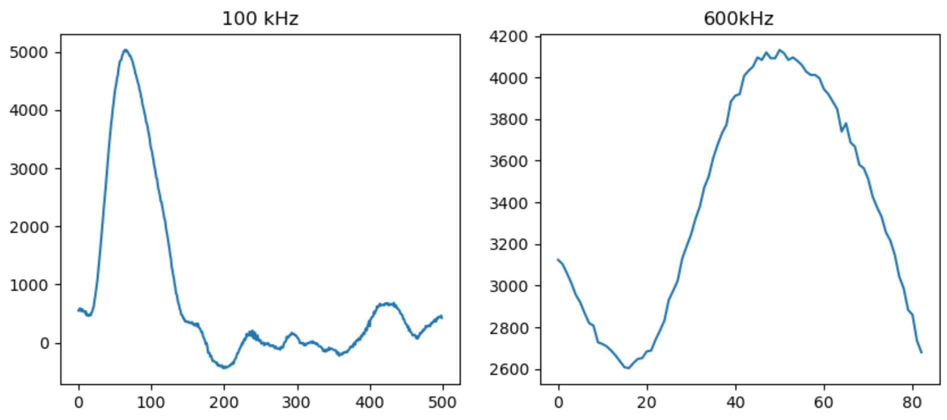


Figure 1: comparison of 100kHz and 600kHz data, where the tail of the previous trace is clearly seen.

At these high repetition rates, traditional methods of classifying the signals by photon number start to fail, such as computing the inner product of the average trace with all traces to produce a histogram (‘stegosaurus‘) where the peaks can be binned and a photon number distribution gained. Other analytic methods such as tail subtraction can perform well up to 600kHz but only for low power, this is because it requires a clean histogram that can be fit and processed, as seen in Fig 2.

Machine learning can be used to classify traces beyond this threshold as it processes only individual traces. Two approaches could be considered, a supervised learning approach where an artificial dataset of labelled traces are produced for training and then tested on the real traces, and an unsupervised approach such as a clustering algorithm. For supervised learning, multiple models could perform well at this task, Deep learning approaches such as recurrent neural networks (RNN) and Convolutional neural networks (CNN) have performed well for a broad range of similar tasks such as electrocardiogram and Supernovae signal classification [3, 4]

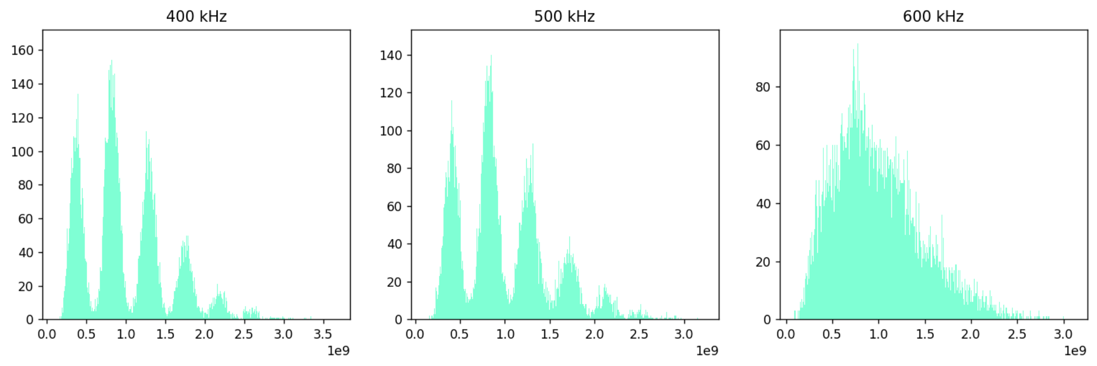


Figure 2: 'Stegosaurus' for $|\alpha|^2 = 1.34$, the overlapping tails dominate the histogram above 500kHz. Tail subtraction requires that each peak is fit with a Gaussian using the *curve_fit* function in the *scipy* package.

For some datasets, tabular classifiers have also been seen to perform well and even outperform deep learning [5]. For tabular classifiers, a feature extraction is required to capture both the temporal and spatial characteristics of the trace, whereas Deep learning methods are able to learn relationships between data points so can be trained on raw time series, similarly with other 'state of the art' methods. Tabular classifiers can sometimes also perform well on raw time series, however this usually due to very strong spatial properties[6].

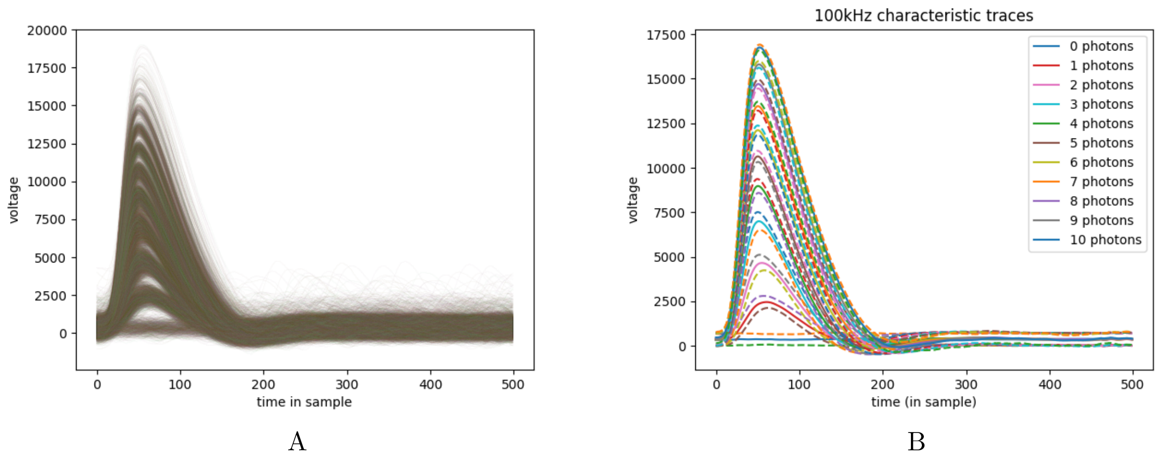


Figure 3: A shows all the traces for 100kHz, plot with a small line width to emphasise the density of the distribution. B shows the characteristic traces and their respective standard deviations

Overall data was taken for reprates ranging from 100kHz to 1MHz and at four different average photon number ($|\alpha|^2$): $|\alpha|^2 = 1.34$, 3.30, 6.03 and 8.14, the maximum photon number recorded was 16. Around 20,000 traces were taken for each repetition rate, which were then split into training and testing files with a ratio of 80:20. There is a significant class imbalance with the training data as the coherent state has a Poissonian distribution, this imbalance could create a bias towards the majority class however, as there is a large number of traces and the testing and actual data also has a Poissonian distribution this was not corrected.

2 Data processing and feature extraction

2.1 Artificial training data

At 100kHz, the traces can be classified exactly by using the histogram to bin traces. This is then filtered and can be used to generate an artificial dataset for model training. This is done by truncating the 100kHz traces to the new period and then adding the tail of the previous trace. Apart from an offset, which is removed before classification, this appears to be a good representation of the data obtained from the TES.

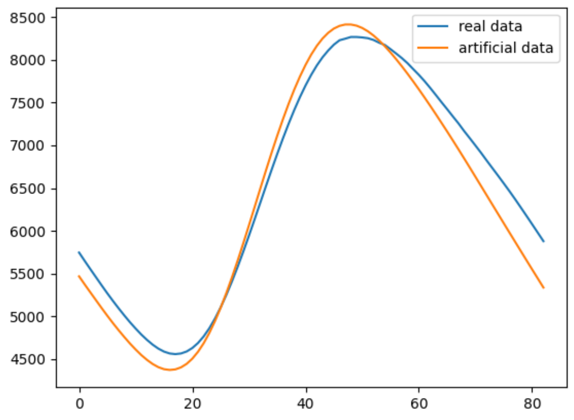


Figure 4: Average traces for both the artificial and real data, data for 600kHz and $|\alpha|^2 = 3.3$

2.2 Feature extraction

The current method of feature extraction is done by calculating nine key features such as peak height, rise time and kurtosis and then returning a feature vector, $F = [f_0, f_1, f_2, ...]$, which would provide the model with all the significant information to classify by photon number.

Table 1: Feature Extraction

Feature	**Formula**
Peak Location	argmax(X)
Average	$\text{average} = \frac{1}{n}\sum_{i=1}^{n} x_i$
Standard Deviation	$\text{std} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \text{average})^2}$
Energy	$\text{energy} = \sum_{i=1}^{n} x_i^2$
Dominant Frequency	$\text{freq} = \text{frequencies}\left[\text{argmax}\left(\text{psd}\right)\right]$, where: frequencies are the estimated frequencies from the Welch method, psd is the corresponding power spectral density estimate
Maximum Peak	max(X)
Rise Time	argmax(X) - arg(X/2)
Crest Factor	$\text{crest} = \frac{\text{max\_peak}}{\sqrt{\text{energy}/n}}$
Kurtosis	$\text{kurt} = \frac{\sum_{i=1}^{n}(x_i - \text{average})^4}{\text{std}^4} - 3$

The three most important features were evaluated using *scipy* selectKbest function and the average, energy and maximum peak were found to have the highest correlation with photon number label. One improvement is fitting equation (2) to the curves and using the fit parameters to characterise the curve. These features would be more closely related to the electrical properties of the pulse however fitting this equation accurately became very demanding for above 20,000 traces. Principal component analysis (PCA) components could potentially also be included to capture further information about the signal. The rise time had a low correlation with the label, suggesting the rise times had little variation among different photon number traces and that either the temporal properties of the trace are relatively invariant or other temporal features must be considered.

3 Model performance

Ten models were tested, their definitions can be found in Section 5. In order to compare the various models performances, metrics such as accuracy, precision, recall and F-1 score were calculated.

They are defined as follows;

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

For an equal comparison, the same dataset was used for all models. This was the set of traces at 600 kHz with an $|\alpha|^2 = 3.30$. This was chosen for two reasons, firstly 600kHz is the limit for tail subtraction performance and secondly it is a sufficiently hard classification task so the spread in performance is visible. No significant hyperparameter tuning was performed on any of the models, so this could improve performance across the board. A detailed explanation of each algorithm and its implementation is found in section 5.

3.1 Tabular classifiers

The classifiers were trained and tested on both the raw data and extracted features. The models considered were: Support vector machines (SVM), Random Forest (RF), Boosted Decision Tree (BDT) and K-Nearest Neighbours (KNN). They were all implemented using the *scikit-learn* package.

Model	Accuracy	Precision	Recall	F1-Score
SVM	0.7657	0.7785	0.7657	0.7640
RF	0.9684	0.9685	0.9684	0.9684
BDT	0.9517	0.9541	0.9517	0.9513
KNN	0.7408	0.7452	0.7408	0.7400

Table 2: Metrics with feature extraction

Model	Accuracy	Precision	Recall	F1-Score
SVM	0.9769	0.9770	0.9769	0.9768
RF	0.9744	0.9745	0.9744	0.9744
BDT	0.9693	0.9694	0.9693	0.9693

Table 3: Metrics without feature extraction

This jump in accuracy suggests the feature extraction is not capturing all the significant information about the trace, so more and better features could be added to improve this. The strong performance with no feature extraction signifies that the spatial properties are significant, and weaker temporal properties. The RF and BDT would recognise these spatial properties such as amplitude and shape patterns in the time series, but not explicitly temporal features such as rise time, which appears to be significant enough for accurate classification.

3.2 Unsupervised Machine learning

Clustering and supervised learning are two different types of machine learning tasks, and they have distinct objectives and evaluation methods. Therefore, they use different metrics to assess their performance. The main goal of clustering is to group similar data points together in an unsupervised manner, without any prior knowledge of the true labels. Clustering evaluation metrics measure how well data points within a cluster are similar and how distinct clusters are from each other. These metrics assess the quality of the clustering structure without using ground truth labels.

The following metrics were used :

- Adjusted Rand Index (ARI): ARI measures the similarity between the true labels and the predicted clusters, accounting for chance. It ranges from -1 to 1, where 1 indicates perfect clustering, 0 indicates random clustering, and negative values indicate worse than random.
- Normalized Mutual Information (NMI): NMI quantifies the mutual information between the true labels and predicted clusters, normalized by entropy. It ranges from 0 to 1, where 1 indicates perfect clustering and 0 indicates no agreement.
- Homogeneity: Homogeneity measures the extent to which each cluster contains only members of a single true class. It ranges from 0 to 1, where 1 indicates perfect homogeneity and 0 indicates poor homogeneity.

Algorithm	**ARI**	**NMI**	**Homogeneity**
K-means	0.556	0.617	0.658
DBSCAN	0.374	0.566	0.577

Table 4: Clustering Evaluation Metrics

The results of these clustering methods indicate it may not be the best method for this task. This may be due to the overlapping tails laying in different cluster zones leading to a lower homogeneity. However one possible improvement is to cluster over a certain part of the trace, such as the ≈ 30 data points surrounding the peak.

3.3 Deep Learning

The two models considered were the RNN and CNN. The RNN was implemented using the *tsai* package and was trained for 50 epochs until the accuracy started to fall, RNN's are specifically designed for time series and work well in understanding the relationships between sequential data points which could process the underlying temporal features of the trace. The most common use of CNN's is image classification, however they can also perform well with time series, the CNN was run for a maximum of 250 epochs where the accuracy and loss were monitored and the learning rate adjusted as required. The best performing model was then chosen for testing. The CNN was implemented using the *tensorflow* package with two convolutional layers.

Model	Accuracy	Precision	Recall	F1-Score
RNN	0.9800	0.9800	0.9800	0.9800
CNN	0.9770	0.9770	0.9770	0.9770

Table 5: Performance Metrics for Deep learning models

The accuracy for both models is very high, with the RNN slightly outperforming the CNN.

3.4 'State of the art'

Time series classification is an active area of research with new algorithms being published frequently. Two leading algorithms are InceptionTime and ROCKET, which were found to lead on standardised datasets [7]. They are both specifically designed for time series but use different approaches, with InceptionTime taking a deep learning approach and ROCKET transforming the data using convolutional kernels then applying a tabular classifier. InceptionTime was trained for ten epochs, and ROCKET for just one and both were implemented using the *tsai* package.

Model	Accuracy	Precision	Recall	F1-Score
InceptionTime	0.9779	0.9782	0.9779	0.9775
Rocket	0.9590	0.9590	0.9590	0.9590

Table 6: Performance Metrics for state of the art models

Accuracy for both models could potentially be improved by training for more epochs however training time is significantly long.

3.5 Summary

Several ML models acheived high accuracies using this dataset, with 6 classifiers with accuracy's $\geq 95\%$, however the time for the models to build and train vary significantly as per fig 5.

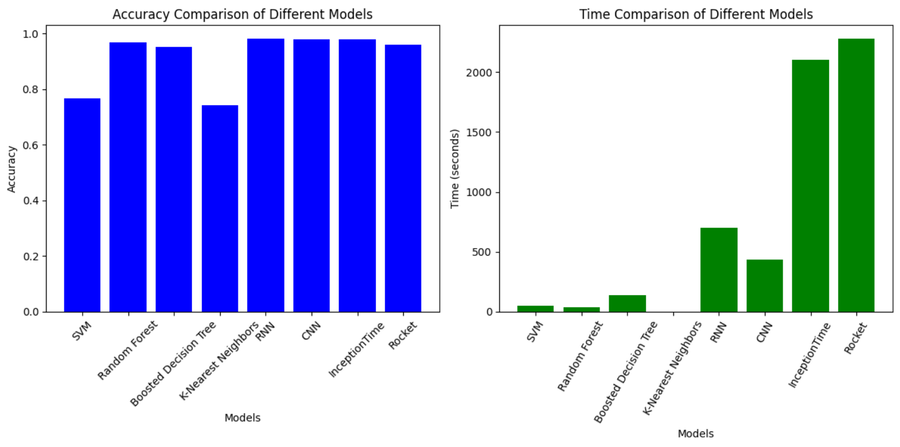


Figure 5: Accuracy scores and respective train/build times for supervised models

The RNN and InceptionTime achieve the highest accuracy, however their speed limits their practical use, especially with datasets this large. If the goal is to have a classifier that functions in real time, such as in [8], this could be limiting unless there was a pre trained model. With a pre-trained model, which would require a 100kHz dataset at the same power, the prediction times are $\geq$ 80 seconds. With accuracies having little variation, the model used depends on whether accuracy or speed is more important. It is also unclear how the performance of different models changes with repetition rate and power.

4 Performance on real data

4.1 ML models

Accuracy scores cannot be calculated for real high rep-rate data, as they have no labels. An alternate metric is to fit the well-classified 100kHz photon number distribution, and then calculate the chi-square statistic between this fit and the photon number distribution produced by classifying the higher rep-rate traces. There is an expected difference between the two distributions but for a large amount of traces, the dominant error is from mis-classification.

Chi-square values were calculated for all rep-rates and models for $|\alpha|^2 = 1.34$, the expected runtime for the two state of the art models and the RNN would total over 14 hours, so only the CNN and the tabular models with no feature extraction were compared for all reprates. It is expected however, based on the model accuracy, that the RNN and InceptionTime would perform slightly better than the CNN in this plot.

As the repetition rate increases, the quality of the fit tails off beyond 800kHz, this is for two main reasons, one is the inability of the training data to accurately replicate the testing dat, which could be improved by adding more overlapping tail into the signals. Secondly the data itself is more difficult to classify. All the models perform indistinguishably well for lower rep-rates, then the SVM starts to fail first, leaving the RF and CNN to be the two strongest performers for higher repetition rates. The 'state of the art' models and the RNN also need to also be evaluated to give a conclusive evaluation.

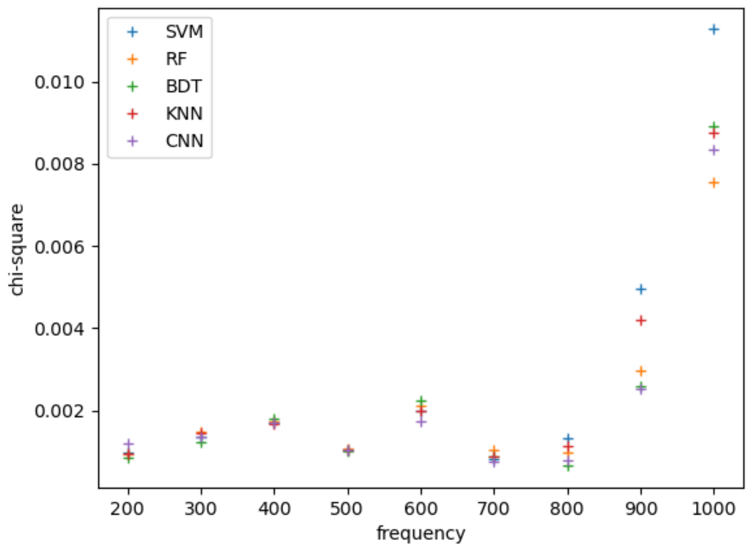
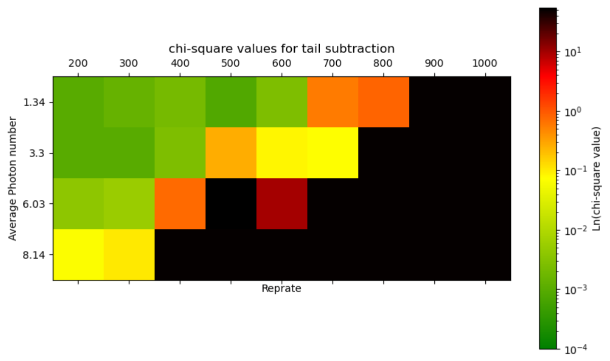


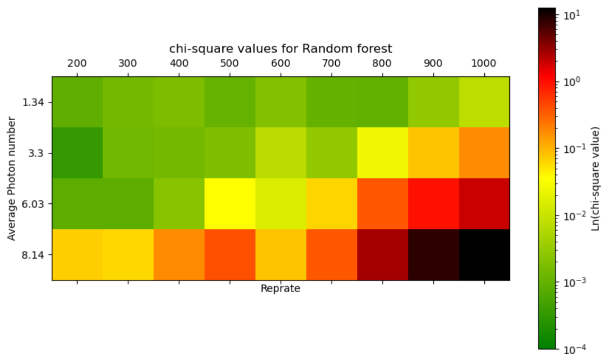
Figure 6: Chisquare values for the $|\alpha|^2 = 1.34$ dataset, the

4.2 Comparison to tail subtraction

Taking the RF as the best model as it has the lowest average chi-square across all frequencies and also relatively fast, it was compared to tail subtraction over all frequencies and powers.



Tail Subtraction



Random Forest

Figure 7:

The chi-square values had a significant range so they were logged for comparison. The RF outperforms tail subtraction for both increasing power and repetition rate, however they both display a similar trend of decay. The decay rate is very different for both models however, Tail subtraction has a very steep decay whereas the RF is more gradual.

5 Model definitions

5.1 Tabular classifiers

The following standard algorithms assume a dataset with feature vectors $X = \{x_1, x_2, \dots, x_n\}$ and corresponding labels $Y = \{y_1, y_2, \dots, y_n\}$, where n is the number of feature vectors:

5.1.1 Support vector machines (SVM)

1. Choose a kernel function (e.g., linear, polynomial, radial basis function) and a regularization parameter C.
2. Formulate the optimization problem to find the hyperplane that maximizes the margin between classes:

$$\begin{aligned} &\text{Minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{subject to} \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad i = 1, 2, \ldots, n \\ &\quad \xi_i \geq 0, \quad i = 1, 2, \ldots, n, \end{aligned}$$

   where w is the weight vector, b is the bias term, ξ_i are slack variables, and y_i are class labels.
3. Solve the optimization problem using techniques like quadratic programming to find the optimal w and b.
4. To classify a new data point x_{new}:
   (a) Compute the decision function $f(x_{\text{new}}) = w \cdot x_{\text{new}} + b$.
   (b) If $f(x_{\text{new}}) > 0$, predict the positive class; if $f(x_{\text{new}}) < 0$, predict the negative class.
5. SVM can be extended for multi-class classification using methods like one-vs-one or one-vs-all.
6. For non-linearly separable data, use the kernel trick to map the data to a higher-dimensional space where separation is possible.
7. SVM aims to find the hyperplane with the largest margin between classes, providing robust generalization. Regularization parameter C balances margin maximization and error minimization.

5.1.2 Boosted Decision Tree

1. Initialize the model by assigning equal weights to all data points: $w_i = \frac{1}{n}$ for $i = 1$ to n.
2. For each boosting iteration $t = 1$ to T:
   (a) Train a decision tree T_t on the weighted dataset (X, Y, w).
   (b) Compute the weighted error ϵ_t of the tree:

$$\epsilon_t = \sum_{i=1}^{n} w_i \cdot I(\hat{y}_i \neq y_i),$$

   where $\hat{y}_i$ is the prediction of T_t for data point x_i, and I is the indicator function.
   (c) Compute the stage value α_t for the current tree:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right).$$

   (d) Update the weights for the next iteration:

$$w_i \leftarrow w_i \cdot \exp\left(-\alpha_t \cdot y_i \cdot \hat{y}_i\right),$$

   where $\hat{y}_i$ is the prediction of T_t for data point x_i.
3. For a new data point x_{new}:
   (a) Let each boosted tree predict the label.
   (b) Aggregate the predictions by weighted majority voting (classification) or weighted averaging (regression) to get the final prediction for x_{new}.

Boosted Decision Trees combine multiple decision trees to create a strong predictive model. Weights are adjusted iteratively to focus on misclassified instances, improving performance. The final prediction is an aggregation of predictions from individual trees.

5.1.3 Random Forest

1. For each tree T_i in the forest ($i = 1$ to N):
   (a) Sample a bootstrapped dataset $(X_{\text{sample}}, Y_{\text{sample}})$ from (X, Y) with replacement. Let the number of samples in the bootstrapped dataset be n_{sample} (equal to n on average).
   (b) Train a decision tree T_i on $(X_{\text{sample}}, Y_{\text{sample}})$ using a random subset of features at each node split. The tree is grown until a stopping criterion is met (e.g., maximum depth or minimum samples per leaf).
2. 2. To make a prediction for a new data point x_{new}:
   (a) Let each tree in the forest predict the label for x_{new}. Denote the predicted label of the i-th tree as $\hat{y}_i$.
   (b) Aggregate the predictions, e.g., by majority voting (classification) or averaging (regression), to get the final prediction for x_{new}. For classification, $\hat{y}_{\text{final}}$ could be:

$$\hat{y}_{\text{final}} = \arg\max_y \sum_{i=1}^{N} I(\hat{y}_i = y),$$

   where I is the indicator function.

The Random Forest algorithm leverages the diversity of multiple trees to improve generalization performance and mitigate overfitting. Each tree is trained on a bootstrapped dataset, and their predictions are combined to make robust and accurate predictions for new data points.

5.1.4 K-Nearest Neighbors (KNN)

1. For a new data point x_{new} for which we want to predict a label:
   (a) Compute the distance between x_{new} and all data points in X. A common distance metric is the Euclidean distance:

$$\text{Distance}(x, x') = \sqrt{\sum_{j=1}^{d} (x_j - x'_j)^2},$$

   where d is the number of features.
   (b) Select the k data points from X that are closest to x_{new} based on distance.
2. For classification:
   (a) Count the occurrences of each class label among the k nearest neighbors.
   (b) Assign the class label with the highest count as the predicted label for x_{new}.
3. For regression:
   (a) Compute the average or weighted average of the target values of the k nearest neighbors.
   (b) Assign this average as the predicted value for x_{new}.

The choice of k is a hyperparameter. A smaller k value makes the algorithm sensitive to noise, while a larger k value may smooth over fine-grained patterns.

5.2 Unsupervised Machine learning

5.2.1 k-means clustering

K-means clustering is an iterative algorithm that aims to partition a dataset into clusters by minimizing within-cluster variance. The algorithm proceeds as follows:

1. **Initialization**: Randomly select k initial centroids from the dataset, representing cluster centers.
2. **Assignment Step**: Calculate the distance of each data point to every centroid and assign it to the nearest centroid based on Euclidean distance.

3. **Update Centroids**: Recalculate the centroids of each cluster by taking the mean of the data points assigned to that cluster.

4. **Repeat**: Iterate the assignment and centroid update steps until convergence or a predefined maximum number of iterations.

While K-means effectively identifies cluster centers, it has limitations. It can converge to local optima due to initial centroid selection. It might struggle with clusters of irregular shapes or varying densities.

5.2.2 DBSCAN

DBSCAN is a density-based clustering algorithm that discovers clusters based on data density. The algorithm proceeds through the following stages:

1. **Core Points**: For each data point, if at least a predetermined points fall within a distance of*epsilon*, the point is marked as a core point.

2. **Cluster Formation**: Starting from a core point, the algorithm forms a cluster by including reachable points within distance *epsilon*. The cluster grows iteratively, consisting of connected core points.

3. **Border Points**: Points reachable from core points but lacking the minimum number of neighbors are classified as border points.

4. **Noise Points**: Points that are neither core nor reachable from any core point are labeled as noise points.

5.3 Deep Learning

5.3.1 Convolutional neural network

1. **Model Initialization:**

   The CNN model is initialized using the Sequential API, which allows for a linear stack of layers. The model will consist of multiple layers that process the input data sequentially.

2. **Convolutional Layers:**

   Two convolutional layers are added to the model, These layers apply 1D convolutions to the input data. The first convolutional layer has 16 filters, a kernel size of 11, "same" padding, and a ReLU activation function. The second convolutional layer is similar but does not require the `input_shape` parameter, as it follows the first layer.

3. **Batch Normalization and Activation:**

   After each convolutional layer, a batch normalisation is applied to normalize the activations. This helps stabilize training. Following batch normalization, a Rectified Linear Unit (ReLU) activation function is applied to introduce non-linearity.

4. **Global Average Pooling:**

   The pooling layer reduces the spatial dimensions of the data. It computes the average value of each feature map over all time steps, resulting in a fixed-length representation.

5. **Fully Connected Output Layer:**

   A fully connected (dense) layer is then added to the model. This layer has a number of units corresponding to the number of classes in the classification problem. The softmax activation function converts the output into a probability distribution over the classes.

6. **Model Compilation:**

   The model is compiled with the Adam optimizer with a learning rate of 0.01. The loss function is categorical cross-entropy, appropriate for multiclass classification.

5.3.2 Recurrent Neural Network

The model architecture used is a LSTM (Long Short-Term Memory) layer followed by a Dense layer.

1. **Model Architecture Setup**: The given model is built using the Sequential API, allowing us to stack layers on top of each other in a sequential manner.
2. **LSTM Layer**: The core component of this model is the LSTM layer. LSTM is a specialized type of RNN that addresses the vanishing gradient problem, making it well-suited for capturing patterns and dependencies in sequences of data. In this layer, there are 64 LSTM units. These units maintain a memory state that can capture information over varying time intervals, allowing the model to remember important context from earlier parts of the sequence.
3. **Dense Layer**: Following the LSTM layer is a Dense layer. This layer is a fully connected neural network layer. The activation function used was the softmax function, which computes a probability distribution over the different classes. This allows the model to output the likelihood of the input sequence belonging to each class.

5.4 'State of the art'

5.4.1 InceptionTime

InceptionTime is a modern deep learning architecture designed for time series classification [9]. It leverages Inception modules, each containing parallel convolutional filters with varying kernel sizes. These filters capture diverse temporal patterns. The process involves:

1. **Convolution:** Input time series undergo convolution with filters of different sizes, extracting local patterns at various scales.
2. **Concatenation:** Outputs of parallel filters are combined, preserving multi-scale information.
3. **Pooling:** Aggregated features pass through pooling, capturing broader temporal characteristics.
4. **Fully Connected:** Flattened features are fed through fully connected layers for classification.
5. **Output:** The final layer employs softmax to predict class probabilities.

This was implemented using the *tsai* package [10]

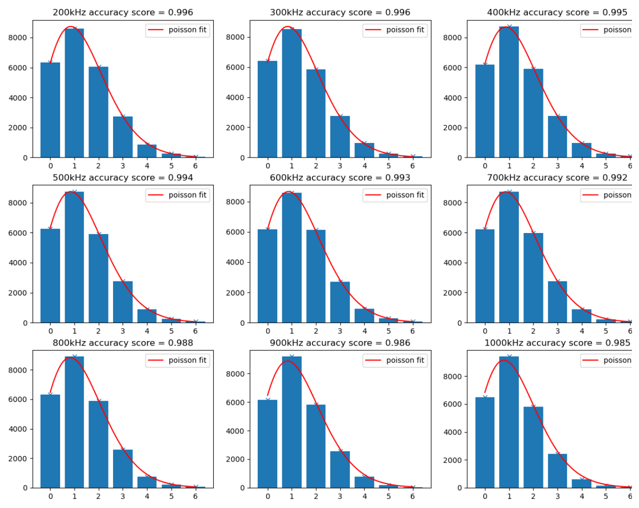
5.4.2 ROCKET

The **ROCKET** (Random Convolutional Kernel Transform) classifier is a modern architecture for time series classification tasks [11]. It efficiently transforms time series data into a higher-dimensional space where classification becomes easier. The process involves:

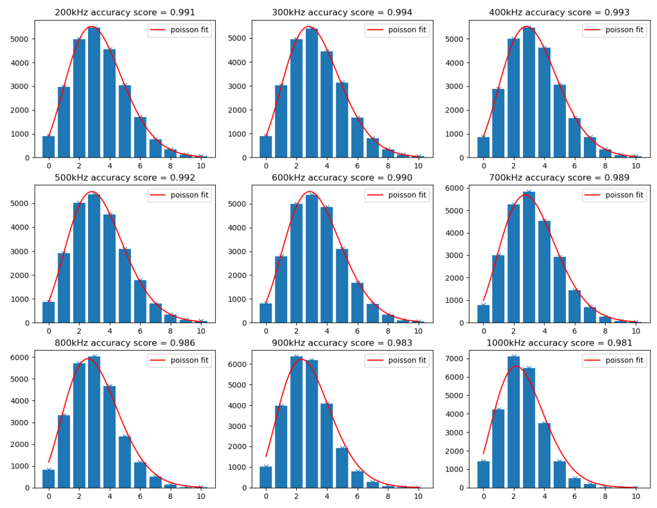
1. **Random Convolutional Kernels:** ROCKET employs a set of randomly generated convolutional kernels. These kernels are applied to the input time series data in a convolutional manner. Each kernel captures different patterns or features from the data.
2. **Convolutional Transform:** The input time series data is convolved with each random kernel. This results in a transformed representation of the data for each kernel. These transformed representations capture diverse aspects of the time series.
3. **Aggregation of Transformations:** The transformed representations from different kernels are aggregated. This aggregation can involve simple statistical measures such as mean, variance, or max pooling. This step helps in extracting informative features from the transformed data.
4. **Classification Layer:** The aggregated features are passed through a classifier, such as a linear SVM or logistic regression. This classifier makes the final decision about the class label based on the extracted features.

6 Extra plots

Photon number distribution for RF with no FE :

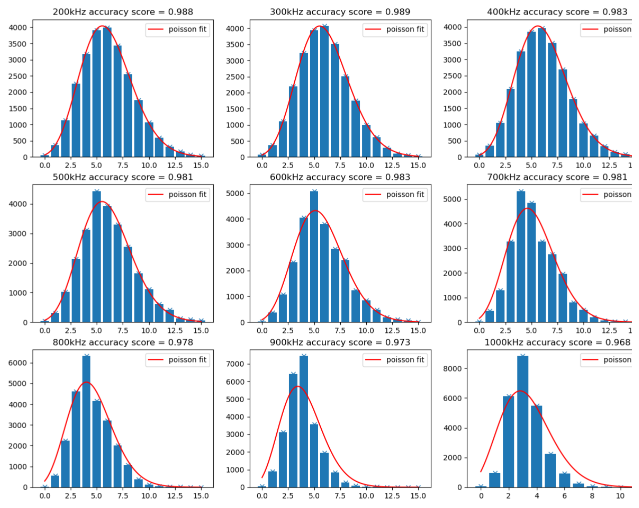


$|\alpha|^2 = 1.34$

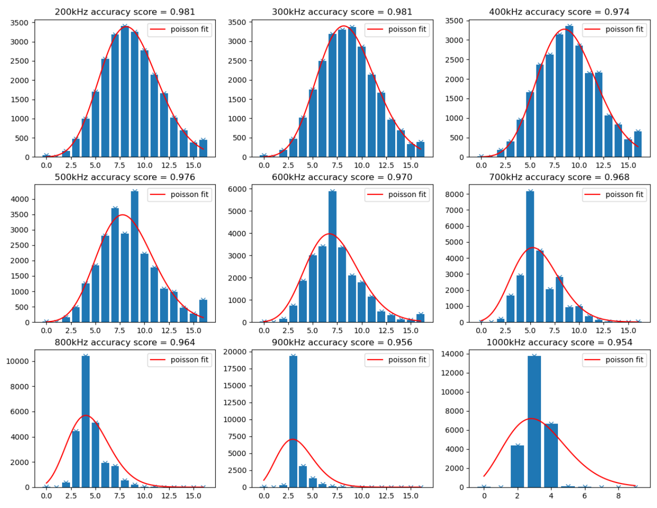


$|\alpha|^2 = 3.30$

Figure 8:



$|\alpha|^2 = 6.03$



$|\alpha|^2 = 8.14$

Figure 9:

References

[1] K. D. Irwin and G. C. Hilton. Cryogenic particle detection. page 63, 2005.

[2] M. Meyer, K. Isleif, F. Januschek, A. Lindner, G. Othman, J. A. Rubiera Gimeno, C. Schwemmbauer, M. Schott, and R. Shah. A first application of machine and deep learning for background rejection in the alps ii tes detector. *ANNALEN DER PHYSIK*, 2023:2200545, 2023.

[3] Shraddha Singh, Saroj Kumar Pandey, Urja Pawar, and Rekh Ram Janghel. Classification of ecg arrhythmia using recurrent neural networks. *Procedia Computer Science*, 132:1290–1297, 2018.

[4] Anthony Brunel, Johanna Pasquet, Jérôme Pasquet, Nancy Rodriguez, Frédéric Comby, et al. A cnn adapted to time series for the classification of supernovae. In *EI 2019 - IS&T International Symposium on Electronic Imaging*, pages 90–1–90–9, 2019.

[5] Maria Frizzarin, Giulio Visentin, Alessandro Ferragina, Elena Hayes, Antonio Bevilacqua, Bhaskar Dhariyal, Katarina Domijan, Hussain Khan, Georgiana Ifrim, Thach Le Nguyen, et al. Classification of cow diet based on milk mid infrared spectra: A data analysis competition at the "international workshop on spectroscopy and chemometrics 2022". *Chemometrics and Intelligent Laboratory Systems*, 234:104755, 2023.

[6] B. Dhariyal, T. Le Nguyen, and G. Ifrim. Back to basics: A sanity check on modern time series classification algorithms. *arXiv e-prints*, 2023.

[7] M. Middlehurst, P. Schäfer, and A. Bagnall. Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *arXiv e-prints*, 2023.

[8] L. A. Morais, T. Weinhold, M. P. de Almeida, J. Combes, A. Lita, T. Gerrits, S. W. Nam, A. G. White, and G. Gillett. Precisely determining photon-number in real-time. 2020.

[9] Ismail Fawaz, Hassan Lucas, Benjamin Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 2020.

[10] Time Series AI. tsai.

[11] A. Dempster, F. Petitjean, and G. I. Webb. Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels. *arXiv e-prints*, 2019.