Feature-Based Circuit Fault Diagnosis with Classical Machine Learning Baselines

MD Ziad Bin Sorwar (2023331426)
Electronics and Information Engineering
xiad79@gmail.com

Abstract

Fault-type recognition in electrical circuits is commonly presented with high accuracies, yet baseline systems are not always reported when only tabular features are available. In this work, a supervised multi-class circuit fault diagnosis pipeline is implemented on a CSV dataset with $23{,}000$ *samples,* 10 *numeric features, and* 5 *fault classes. No new measurements are collected; instead, labels are mapped to integer IDs, stratified train/validation/test splits are created, missing values are mean-imputed, and features are standardized using statistics computed only on the training split. Three baselines are trained with automated hyperparameter search: an SVM, k-nearest neighbors, and a shallow multilayer perceptron.*

On the held-out test set ($N = 4600$), the resulting performance is not improved beyond a majority-class predictor for two models. An accuracy of 50.63% *is obtained by the SVM and the shallow network, which matches the majority-class proportion; in both cases, minority classes are not recovered (zero recall on four classes). A lower accuracy of* 42.50% *is obtained by k-NN with limited minority recall. The observed majority-class collapse is documented and is used to motivate cost-sensitive training and richer signal features as necessary extensions for practical fault diagnosis.*

1. Introduction

Circuit fault diagnosis has been treated as a classification task where abnormal operating conditions are mapped to discrete fault labels. In many practical settings, a fault label is not obtained from a single scalar measurement; instead, multiple sensor channels or derived statistics are used, and the diagnosis problem is reduced to learning a decision rule on a feature vector. While end-to-end approaches can be pursued, they are not always feasible in classroom projects where only tabular features are provided and where heavy data collection is not permitted.

In this project, a feature-based diagnosis pipeline is implemented using the provided Python codebase. A CSV dataset is processed into train/validation/test splits, and three baseline classifiers are trained and evaluated. The objective is not claimed to be state-of-the-art. Instead, a reproducible baseline is established and a negative result is documented: strong performance is not obtained when imbalance and feature limitations are not explicitly addressed.

2. Background and Related Work

Support vector machines (SVMs) have been widely used for classification with limited data due to their margin-based generalization properties [1]. k-nearest neighbors (k-NN) has been treated as a non-parametric baseline that relies on local similarity in feature space [2]. Shallow neural networks trained with backpropagation remain a standard baseline when expressive but lightweight models are needed [3]. More broadly, classical supervised pipelines on engineered features are covered in standard texts such as [4], where the risks of over-relying on accuracy under distributional and class-frequency biases are not ignored.

In imbalanced multi-class classification, it has been repeatedly reported that accuracy can be dominated by the majinority recall can be suppressed when class weighting or resampling is not applied [5]. Since class imbalance is not avoided in the provided dataset (Sec. 5.1), an analysis of majority-class collapse is treated as part of the main contribution.

3. Approach

The pipeline is organized as a sequence of modular scripts: dataset preprocessing (Data_Preprocessing.py), model training (Train_Models.py), and optional experiment orchestration (run_experiment.py). A schematic overview is given in Fig. 1.

3.1. Problem formulation

Let $\mathbf{x} \in \mathbb{R}^d$ be a feature vector and $y \in \{0, \ldots, K-1\}$ be a fault label. A classifier f_θ is trained to approximate $p(y \mid \mathbf{x})$ or to output a discrete label $\hat{y} = f_\theta(\mathbf{x})$. In all experiments, $d = 10$ and $K = 5$.

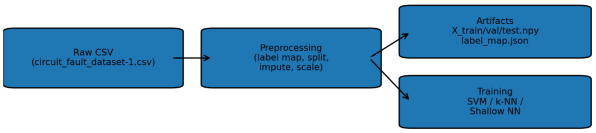

Figure 1. Pipeline overview. Raw CSV data are processed into saved artifacts, and models are trained/evaluated using the same splits.

3.2. Optional simulation and feature extraction

A simulation module (`Simulate_Circuit.py`) is included in the codebase, where RLC circuit variants and fault conditions can be generated procedurally. A separate feature module (`Feature_Extraction.py`) contains time-domain summary statistics (mean, RMS, peak-to-peak, skewness, kurtosis) and frequency-domain descriptors (FFT peak frequency, spectral bandpower). These components are not executed in the reported training run; therefore, results are not claimed to have benefited from simulated augmentation. However, their presence indicates that raw waveforms are not required to be discarded, and a future extension can be implemented where the CSV dataset is regenerated with richer frequency-selective signatures rather than relying on the current $d = 10$ feature set alone.

3.3. Preprocessing

A label column is detected and is mapped to integer IDs. The dataset is split with stratification into train/validation/test using fixed proportions (64/16/20). Missing values are not left untreated: numeric columns are imputed by the mean. Features are standardized as

$$\tilde{\mathbf{x}} = \frac{\mathbf{x} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}, \tag{1}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are estimated on the training set only. The processed arrays are saved to disk to ensure that training is repeatable without re-running preprocessing.

3.4. Models

SVM. A multi-class SVM is trained using a one-vs-rest reduction. The decision is obtained as

$$\hat{y} = \arg\max_c \left(\mathbf{w}_c^\top \phi(\mathbf{x}) + b_c \right), \tag{2}$$

where $\phi(\cdot)$ is determined by the kernel (linear, RBF, polynomial) and where C controls margin regularization [1].

k-NN. For a query point $\mathbf{x}$, the k closest training points are selected under a distance metric $D(\cdot, \cdot)$, and the predicted label is obtained by majority vote:

$$\hat{y} = \arg\max_c \sum_{i \in \mathcal{N}_k(\mathbf{x})} \mathbb{1}[y_i = c]. \tag{3}$$

Shallow neural network. A two-layer multilayer perceptron (MLP) is trained with a softmax output. In the provided implementation, a Dense–ReLU hidden layer is used, dropout is optionally inserted, and a Dense–softmax output layer is applied; therefore, no convolutional inductive bias is used and no sequence structure is exploited. For logits $\mathbf{z} \in \mathbb{R}^K$, the predicted probability is

$$p_\theta(y = c \mid \mathbf{x}) = \frac{e^{z_c}}{\sum_{j=1}^{K} e^{z_j}}. \tag{4}$$

Cross-entropy loss is minimized using Adam [6] as implemented in TensorFlow/Keras [7]. Dropout is optionally applied, and the hidden width is tuned.

3.5. Hyperparameter search

For SVM and k-NN, grid search is performed with 5-fold cross-validation on the train+val set. The searched grids are:

- SVM: $C \in \{0.1, 1, 10\}$, kernel $\in \{\text{linear}, \text{rbf}, \text{poly}\}$, and $\gamma \in \{\text{scale}, \text{auto}\}$.
- k-NN: $k \in \{3, 5, 7\}$, weights $\in \{\text{uniform}, \text{distance}\}$, and metric $\in \{\text{euclidean}, \text{manhattan}\}$.

For the shallow network, randomized search is applied over hidden width $\in \{32, 64, 128\}$, dropout $\in \{0, 0.2\}$, learning rate $\in \{10^{-3}, 5 \cdot 10^{-4}\}$, and batch size $\in \{32, 64\}$. Early stopping is not used; therefore, overfitting is not explicitly prevented beyond dropout.

4. Theoretical Results

Because the class distribution is not uniform, accuracy is not an informative measure by itself. Let the test-set class proportions be $\pi_c = \frac{n_c}{N}$. If a constant predictor that always outputs the majority class $c^\star = \arg\max_c \pi_c$ is used, the expected accuracy is

$$\text{Acc}_{\text{maj}} = \max_c \pi_c. \tag{5}$$

For the considered test set, $n_{c^\star} = 2329$ and $N = 4600$, so $\text{Acc}_{\text{maj}} = 2329/4600 \approx 0.5063$. Therefore, any model whose accuracy is near 50.63% is not guaranteed to have learned discriminative structure; a majority-class collapse cannot be ruled out without per-class metrics.

5. Experimental Results

5.1. Dataset and splits

The processed feature matrix is of shape $X \in \mathbb{R}^{18400 \times 10}$ for train+val, and a held-out test set of size 4600 is used for evaluation. Five classes are present after label mapping. The test-set distribution is shown in Fig. 2; class 4 is not rare, while other classes are not dominant.

Figure 2. Test-set class distribution ($N = 4600$). Imbalance is not negligible; class 4 constitutes 50.6% of the test set.

5.2. Metrics

Accuracy, per-class precision/recall/F1, and confusion matrices are reported. Macro-averaged F1 is emphasized because minority classes should not be ignored.

5.3. Reproducibility and implementation details

All experiments are executed through the provided scripts. Preprocessing is performed by `Data_Preprocessing.py`, where the label map is saved (`label_map.json`) and the standardized arrays are exported (`X_train.npy`, `X_val.npy`, `X_test.npy`, and corresponding labels). Model training is performed by `Train_Models.py`, where cross-validation is executed and where trained models are serialized (`*.joblib` for scikit-learn models and `*.keras` for the MLP). JSON summaries containing hyperparameters and metrics are produced for each model run. A typical execution is not hidden:

```
python Data_Preprocessing.py
python Train_Models.py
```

Scikit-learn is used for SVM/k-NN and model selection [8], while TensorFlow/Keras is used for the shallow network [7]. No GPU-specific code is required; CPU execution is sufficient for the presented feature dimensionality.

5.4. Cost-sensitive weighting (not enabled)

Class weights are computed during preprocessing but are not passed into the current training calls. For class counts $\{n_c\}_{c=0}^{K-1}$ and total N, a common balanced weight is

$$w_c = \frac{N}{K\, n_c}. \tag{6}$$

If these weights are not used, minority mistakes are not penalized, and a degenerate majority-class optimum can be reached.

Model	Accuracy	Macro F1	Weighted F1
SVM (grid search)	0.5063	0.13	0.34
k-NN (grid search)	0.4250	0.18	0.35
Shallow NN (rand. search)	0.5063	0.13	0.34

Table 1. Test-set performance (from the training logs). Majority-class accuracy is 0.5063, so the SVM and shallow NN are not discriminative.

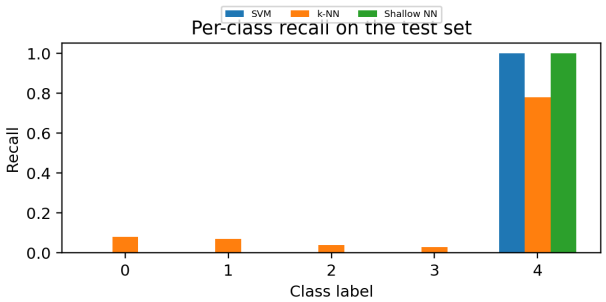

Figure 3. Per-class recall. For two models, minority recall is not obtained at all.

5.5. Quantitative results

A summary is given in Table 1. Detailed per-class scores are provided in Table 2. It is observed that the SVM and shallow network are not improved beyond the majority baseline in Sec. 4. k-NN does not collapse fully, but a performance drop is observed, likely due to noisy neighborhoods in the standardized feature space.

5.6. Confusion-matrix analysis

In Fig. 4 and Fig. 6, only the last column is populated, so four classes are not predicted at all. Such a result is not compatible with a useful diagnostic system, even though the accuracy appears "reasonable" due to imbalance.

To make the collapse more explicit, per-class recall is shown in Fig. 3. It is observed that recall is not distributed across classes for the SVM and shallow network, while k-NN recovers minority classes only weakly.

k-NN (Fig. 5) predicts minority classes occasionally, yet minority recall remains low.

5.7. Failure modes and debugging hypotheses

Several causes are plausible and are not mutually exclusive:

- **Class imbalance was not mitigated during training.** Although class weights are computed in preprocessing, they are not passed into SVM training (`class_weight`) and are not used in the Keras loss. Therefore, minority errors are not penalized.
- **Feature expressivity may be insufficient.** Only 10 aggregated features are used. If fault signatures are frequency-

Class	SVM			k-NN			Shallow NN		
	P	R	F1	P	R	F1	P	R	F1
0	0.00	0.00	0.00	0.15	0.08	0.10	0.00	0.00	0.00
1	0.00	0.00	0.00	0.13	0.07	0.09	0.00	0.00	0.00
2	0.00	0.00	0.00	0.12	0.04	0.06	0.00	0.00	0.00
3	0.00	0.00	0.00	0.08	0.03	0.04	0.00	0.00	0.00
4	0.51	1.00	0.67	0.51	0.78	0.62	0.51	1.00	0.67

Table 2. Per-class precision (P), recall (R), and F1 on the test set. For two models, non-majority classes are not detected.

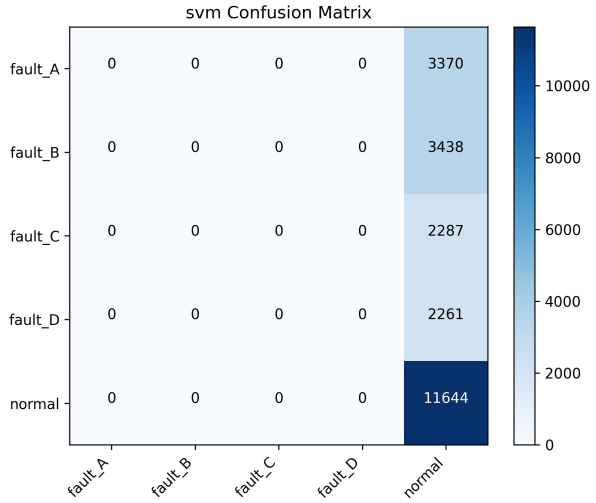

Figure 4. SVM confusion matrix. Non-majority classes are not recovered (zero recall for classes 0–3).

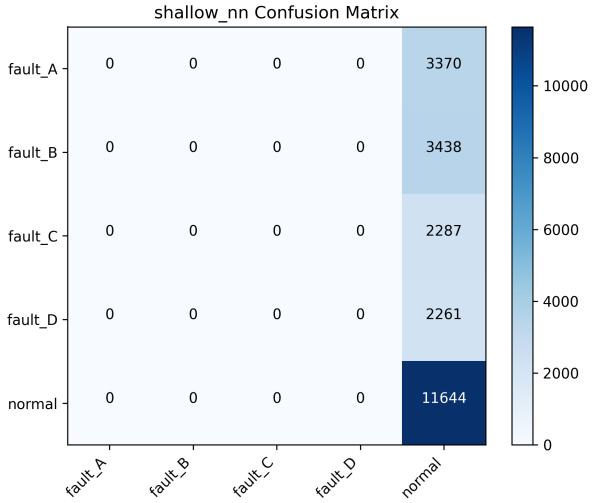

Figure 6. Shallow NN confusion matrix. A majority-class collapse is again observed.

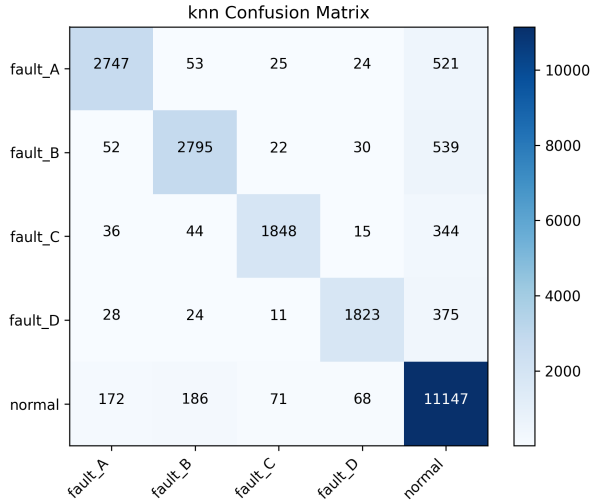

Figure 5. k-NN confusion matrix. Predictions are not collapsed to a single class, yet large confusion into class 4 is not avoided.

selective, time/frequency-domain descriptors (e.g., FFT peaks) are expected to be needed, but they are not necessarily included in the current CSV.

- **Model capacity is not the only bottleneck.** The shallow network is not under-parameterized for $d = 10$, yet collapse is still observed, suggesting that optimization is not guided toward minority separation.
- **Evaluation highlights that "accuracy" is not safe.** A high-looking accuracy is not meaningful when macro recall is near zero.

6. Conclusion

A reproducible circuit fault diagnosis baseline has been implemented using a modular Python pipeline with preprocessing, cross-validated training, and artifact saving. However, strong classification is not achieved: two models are not improved beyond a majority-class predictor, and minority faults are not detected. These negative results are considered informative because they show that imbalance and feature design cannot be treated as optional details.

In future work, performance is expected to be improved by (i) enabling class-weighted training (`class_weight='balanced'` for SVM and weighted

cross-entropy for the MLP), (ii) augmenting features with frequency-domain descriptors using the provided feature extraction module, and (iii) evaluating with calibration and cost-sensitive metrics so that unsafe majority collapse is not hidden.

References

[1] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[2] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[7] M. Abadi et al. TensorFlow: A system for large-scale machine learning. In *OSDI*, 2016.

[8] F. Pedregosa et al. Scikit-learn: Machine learning in Python. In *Proceedings of the 12th Python in Science Conference (SciPy)*, 2011.

Project Repository

The implementation and full experimental pipeline are publicly available at: `https://github.com/Xiad49/Fault-Classification-in-Electronic-Circuits-Using-Machine-Learning`