Predicting DPE Labels from Open Data

Model Build Report (Paris, 2018–2021)

Open Data Challenge (OpenData University × Enedis)

PBL Year 2

March 4, 2026

Contents

1 Overview

This report documents how we built a machine learning model that predicts the DPE energy label (`etiquette_dpe`, from A to G) from a small and interpretable set of building and energy descriptors available in open DPE data. The motivation is practical: if a DPE record contains structured inputs such as surface, standardized consumption indicators, construction period, and heating energy, then a model should be able to infer a plausible label. This capability is valuable for quality checks (detecting records that look inconsistent) and as a reusable component in a broader product pipeline. In the wider Open Data Challenge project, we also work with Enedis consumption data to connect DPE information to measured electricity; the model described here, however, is focused specifically on predicting the DPE label itself.

2 Data sources and scope

We restricted the analysis to **Paris** and to the period **2018–2021**. DPE data was sourced from ADEME open datasets. Because the DPE methodology changed in mid-2021, ADEME provides two relevant datasets: the pre-2021 dataset (`dpe-france`) and the post-2021 dataset (`dpe03existant`). For the broader project context, we also used Enedis open data on annual residential consumption by address (`consommation-annuelle-residentielle-par-adresse`) and cleaned it to retain Paris-only observations.

3 Target and feature design

The supervised target is the DPE label, `etiquette_dpe`, taking one of seven classes {A, B, C, D, E, F, G}. We deliberately limited the feature set to keep the model lightweight and interpretable. The model uses six raw input variables:

- **Numerical:** `surface_habitable_logement`, `conso_5_usages_par_m2_ep`, `emission_ges_5_usages`
- **Categorical:** `type_batiment`, `periode_construction`, `type_energie_principale_chauffage`

After preprocessing (scaling + one-hot encoding), these inputs become a 24-dimensional vector used by the classifier.

4 Preprocessing pipeline

We implemented preprocessing and modeling as a scikit-learn `Pipeline` to guarantee that the same transformations are applied during both training and inference. Numerical variables are imputed with the median and standardized using `StandardScaler(with_mean=False)`. Categorical variables are imputed with the most frequent category, coerced to strings using a helper function `to_str`, and one-hot encoded using `OneHotEncoder(handle_unknown="ignore")` so that unseen categories do not break inference.

Issue encountered: loading the saved model

When reloading the model saved with `joblib`, we initially hit:

```
AttributeError: Can't get attribute 'to_str' on <module '__main__'>
```

This happened because `to_str` was defined inside the notebook (`__main__`), and joblib pickles a reference to that symbol. The immediate workaround was to define `to_str` before calling `joblib.load`. The robust fix is to move `to_str` into a dedicated Python module (e.g., `preprocessing_utils.py`), import it during training/inference, and re-save the pipeline.

5 Train/test split and dataset sizes

To evaluate fairly under class imbalance, we used a stratified 80/20 split implemented with `StratifiedShuffleSplit` (`test_size=0.2`, `random_state=42`). Stratification preserves the A–G label distribution across training and test sets. For faster iteration, we created a dev subset by sampling up to 15,000 observations per class from the training split.

The resulting dataset sizes were:

- **Dev set:** (80,349, 11)
- **Test set:** (150,155, 11)

Note on leakage risk. This split is stratified but not grouped. If multiple DPE records refer to the same building/address, similar observations could appear in both train and test, inflating performance. If duplication is significant, a grouped split (by building/address identifier) would provide a more conservative estimate.

6 Model choice and training

We trained a linear multiclass classifier using `SGDClassifier` with `loss="log_loss"` (multinomial logistic regression trained by stochastic gradient descent). This is a good fit for sparse one-hot encoded features and keeps the model interpretable. We used `class_weight="balanced"` to reduce bias toward majority classes.

The main hyperparameters were:

- `alpha=5e-06` (regularization strength)
- `max_iter=200`, `tol=1e-4` (optimization)
- `random_state=42`, `n_jobs=-1`

During fitting, scikit-learn warned that the iteration limit was reached before convergence. This does not automatically invalidate the model, but it indicates an improvement opportunity. In practice we would increase `max_iter`, adjust `tol`, and/or slightly increase `alpha` to stabilize the solution and reduce sensitivity to noise.

7 What the model looks like (explicit mathematical form)

A key advantage of this model family is that it can be written as an explicit formula. Let $\phi(x) \in \mathbb{R}^{24}$ be the transformed feature vector after preprocessing. For each class $k \in \{A, \ldots, G\}$, the model computes a linear score:

$$s_k(x) = b_k + \sum_{j=1}^{24} w_{k,j}\, \phi_j(x),$$

and converts the scores into probabilities via the softmax function:

$$P(y = k \mid x) = \frac{e^{s_k(x)}}{\sum_{c \in \{A,\ldots,G\}} e^{s_c(x)}}.$$

The predicted label is $\hat{y} = \arg\max_k s_k(x)$. In our trained artifact, the learned parameter matrix has shape `coef` $\in \mathbb{R}^{7\times 24}$ and the intercept vector has shape `intercept` $\in \mathbb{R}^7$. Because this is multiclass softmax, weights should be interpreted *relative to other classes*: a negative coefficient for a given class does not mean a feature makes that class impossible, only that it contributes less to that class score than it does to competing class scores.

8 Evaluation results

We evaluated on the held-out test set (n = 150,155). Overall accuracy was 0.73, and balanced accuracy (macro-average recall) was 0.798. The macro-averaged F1 score was 0.72, and the weighted F1 score was also 0.72.

Per-class results show that performance is strong on the large-support classes and less stable on rare ones. In particular, class A has extremely low support (43), so its precision/recall estimates are noisy. Classes C and D dominate the dataset (supports 25,859 and 43,876) and are predicted relatively well (F1 of 0.85 and 0.71). Class E is predicted conservatively: precision is high (0.89) but recall is low (0.45), meaning many true E cases are classified as neighboring labels.

9 Limitations and next improvements

The immediate technical improvements are (i) addressing non-convergence by increasing `max_iter` and tuning `alpha`, and (ii) validating robustness with a grouped split if duplicate records exist per building/address. From a modeling perspective, class A scarcity limits how confidently we can evaluate or optimize performance on that label alone; depending on product needs, merging rare labels or focusing on broader risk bands may be more stable.

Finally, in the project roadmap toward the "Reality-Adjusted DPE" product, the next step is to train a regression model on measured electricity consumption (from Enedis) and add uncertainty estimates (prediction intervals). The classifier described here remains useful as an auxiliary signal for consistency checks and interpretability.