IML2025 Term Project: Classifying New Particle Formation

Final Report

December 18, 2025

Team Name: Group 268
Team Members: Viljami Ranta

1 Introduction

New Particle Formation (NPF) is an atmospheric event where gas molecules grow into bigger particles. These particles can eventually affect cloud formation and climate, making NPF prediction valuable for atmospheric science research. The data for this project comes from the SMEAR II station in Hyytiälä, Finland, where continuous measurements of temperature, radiation, humidity, and trace gases are recorded.

The classification task involves two levels: binary classification distinguishing event days (when NPF occurs) from nonevent days, and multi-class classification identifying the event type (Ia, Ib, or II) based on the strength and clarity of the formation event. The competition score combines three components equally: binary accuracy, multi-class accuracy and a perplexity term that rewards good probability estimates.

The dataset has several challenges: only 450 training samples, around 100 features with a lot of redundancy, and severe class imbalance in the multi-class. My approach addresses these challenges through a two-stage classification strategy, first predicting event versus nonevent with Lasso logistic regression, then classifying classes with a separate Ridge model trained only on event samples. This decomposition achieved a final score of 0.769, placing 13th on the public leaderboard.

2 Data Exploration

2.1 Dataset Overview

Let's start by loading the data:

```
df_train = pd.read_csv("./data/train.csv")
df_test = pd.read_csv("./data/test.csv")
```

Let's look at some basic information:

```
print(f"training samples: {len(df_train)}")
print(f"test samples: {len(df_test)}")
print(f"number of features: {df_train.shape[1]}")
print(f"first rows: \n{df_train.head()}")
```

```
training samples: 450
test samples: 965
number of features: 104
first rows:
   id        date    class4  partlybad  CO2168.mean  CO2168.std
      CO2336.mean  ...   T84.std  UV_A.mean   UV_A.std  UV_B.mean
      UV_B.std   CS.mean    CS.std
0   0  2000-03-21        II      False   372.396757    0.752494
   372.279392  ...  2.283825   6.237543   4.372063   0.115203
   0.104295  0.000510  0.000123
1   1  2000-03-23  nonevent      False   372.889867    0.410639
   372.769205  ...  1.979027  11.626868   7.208083   0.301720
   0.229672  0.000706  0.000250
2   2  2000-04-07        Ia      False   373.869464    0.655604
   373.788580  ...  1.929516  16.688892  10.504951   0.561251
   0.451130  0.000851  0.000244
3   3  2000-04-09        Ib      False   376.006588    1.109789
   375.888889  ...  3.161601  17.456796  10.967471   0.716453
   0.572409  0.002083  0.000203
4   4  2000-04-14  nonevent      False   374.068239    1.257096
   374.042330  ...  0.929537   4.279844   2.425409   0.146308
   0.106017  0.002650  0.000891

[5 rows x 104 columns]
```

The training set contains 450 samples and the test set 965 samples, with 104 features.

After removing id, date, class4, and partlybad, we have approximately 100 numeric features to work with.

The test set is roughly twice the size of the training set. This means our model needs to generalize well from pretty small training samples.

Looking at the features, we can identify several groups:

- Temperature (T48, T672, etc.) at various heights
- CO2 concentration (CO2168, CO2336) at different heights
- UV radiation (UV_A, UV_B)
- Condensation sink (CS) - a key variable for particle formation physics

Each measurement appears twice as .mean and once as .std, giving us both the average value and the variability in a day. The std features might capture if conditions were stable, this may be relevant since particle formation might require stable conditions?

The partlybad column flags days with potential data quality issues. I ignored it since the column was not predictive and removing flagged days would reduce the already small dataset.

2.2 Class Distribution

Let's look at the class distribution:

```
print("class4 distribution:")
print(df_train["class4"].value_counts().to_latex())
print("class4 proportion:")
print(df_train["class4"].value_counts(normalize=True).round(3).
   to_latex())
```

class4	count
nonevent	225
II	117
Ib	82
Ia	26

class4	proportion
nonevent	0.500000
II	0.260000
Ib	0.182000
Ia	0.058000

The binary classes are balanced: 225 event and 225 nonevent days.

Multi-class distribution is imbalanced. Class II is most common (117 samples, 26%), followed by Ib (82 samples, 18%), and Ia (26 samples, 5%). With only 5% Ia, the model will probably struggle to learn this class.

The approach will probably be to focus more on the binary classification then consider multi-class.

2.3 Feature Statistics

Let's look at some statictics of the features. I have only included one representative feature from each "group" we identified.

The complete list of these features is provided in appendix A.

```
print("summary of features:")
print(df_train[example_features].describe().T.to_latex(escape=True,
    index=False, float_format="%.3f",))
```

	count	mean	std	min	25%	50%	75%	max
CO2168.mean	450.000	382.077	11.168	359.087	373.872	381.358	389.397	414.864
CO2168.std	450.000	3.352	3.448	0.121	0.975	2.228	4.529	22.822
Glob.mean	450.000	191.993	126.612	3.926	65.420	199.001	298.567	426.457
Glob.std	450.000	141.429	92.283	2.006	41.814	157.012	222.718	318.331
H2O168.mean	450.000	7.173	3.868	1.001	3.937	6.513	9.613	19.148
H2O168.std	450.000	0.568	0.509	0.014	0.212	0.433	0.770	3.872
NET.mean	450.000	121.421	87.462	-59.714	38.392	122.181	196.112	302.606
NET.std	450.000	123.781	82.799	1.635	36.538	130.578	195.968	276.464
NO168.mean	450.000	0.105	0.335	-0.015	0.017	0.037	0.082	5.111
NO168.std	450.000	0.106	0.149	0.021	0.050	0.063	0.093	1.653
O3168.mean	450.000	33.333	10.057	0.954	26.776	33.126	40.557	71.332
O3168.std	450.000	3.640	2.483	0.210	1.720	3.203	4.952	14.468
Pamb0.mean	450.000	991.144	11.057	954.884	983.728	991.672	998.016	1022.559
Pamb0.std	450.000	1.092	0.860	0.070	0.427	0.870	1.452	5.525
PAR.mean	450.000	379.376	248.637	9.967	130.643	396.159	595.232	825.892
PAR.std	450.000	279.864	182.990	4.052	86.542	306.228	440.495	601.128
PTG.mean	450.000	0.001	0.009	-0.008	-0.002	-0.000	0.001	0.091
PTG.std	450.000	0.009	0.007	0.000	0.004	0.008	0.013	0.052
RGlob.mean	450.000	27.439	17.313	-16.465	11.337	28.441	41.784	102.054
RGlob.std	450.000	18.385	10.210	0.557	8.560	21.464	26.494	48.314
RHIRGA168.mean	450.000	69.340	19.769	29.029	53.731	68.498	88.378	115.149
RHIRGA168.std	450.000	8.337	5.752	0.287	2.339	8.746	12.425	24.643
RPAR.mean	450.000	19.658	14.972	0.029	9.055	18.327	25.807	148.944
RPAR.std	450.000	13.740	9.412	0.213	7.349	13.953	17.222	76.968
SO2168.mean	450.000	0.279	0.430	-0.027	0.056	0.126	0.327	4.264
SO2168.std	450.000	0.164	0.146	0.044	0.076	0.110	0.197	1.104
SWS.mean	450.000	906.998	34.559	711.615	909.669	918.323	922.295	932.455
SWS.std	450.000	23.477	41.933	0.000	0.829	2.062	22.132	190.652
T168.mean	450.000	6.356	9.816	-23.618	-0.837	7.560	13.985	26.703
T168.std	450.000	1.834	1.127	0.041	0.815	1.904	2.727	5.246
UV_A.mean	450.000	10.797	6.627	0.439	4.305	11.507	16.573	22.500
UV_A.std	450.000	7.614	4.947	0.121	2.700	8.018	11.832	16.320
CS.mean	450.000	0.003	0.002	0.000	0.001	0.003	0.004	0.014
CS.std	450.000	0.001	0.001	0.000	0.000	0.001	0.001	0.006

Features have very different scales. CO2 average at 382, temperature (T168) around 6, condensation sink (CS) around 0.003. Standardization will be necessary for algorithms like SVM and logistic regression.

Some interesting properties:

- High variability: Glob and PAR have very large STD, indicating highly variable conditions across days
- Near-zero values: PTG.mean is around 0.0005
- Negatives: NET.mean and RGlob.mean can go negative
- Temperature range: T168.mean spans from -24 to +27, capturing the whole climate

2.4 Feature Correlations

Let's look at the correlations between features. I will separately check mean and std features. I will only check the example_features (one per "group") because features in the same group are highly correlated.

The full code for finding the high correlation pairs is in Appendix B.

```python
print("highly correlated feature pairs:")
for pair in high_correlation_pairs:
    print(f"{pair['feature1']} and {pair['feature2']}: {pair['
       correlation']:.3f}")
```

```
highly correlated feature pairs:
Glob.mean and NET.mean: 0.959
Glob.mean and PAR.mean: 0.996
Glob.mean and RHIRGA168.mean: -0.907
Glob.mean and UV_A.mean: 0.985
NET.mean and PAR.mean: 0.968
NET.mean and UV_A.mean: 0.968
PAR.mean and RHIRGA168.mean: -0.912
PAR.mean and UV_A.mean: 0.993
Glob.std and NET.std: 0.983
Glob.std and PAR.std: 0.997
Glob.std and UV_A.std: 0.988
NET.std and PAR.std: 0.984
NET.std and UV_A.std: 0.978
PAR.std and UV_A.std: 0.993
```

The radiation-related features (Glob, PAR, UV_A, NET) are all highly correlated with each other. This makes sense, they all basically measure solar radiation.

Relative humidity (RHIRGA168) shows strong negative correlation with radiation variables. Sunny days are drier, very intuitive.

For modeling, this likely means we don't need to use all of them.

To visualize these relationships more clearly, I created a correlation heatmap focusing on the most strongly correlated features. The heatmap (Figure 1) visually confirms the strong positive correlations within radiation features (Glob, PAR, UV_A, NET) and the negative correlation between radiation and relative humidity.

This visualization helped confirm that many of these features provide redundant information, supporting the decision to rely on Lasso regularization for implicit feature selection.

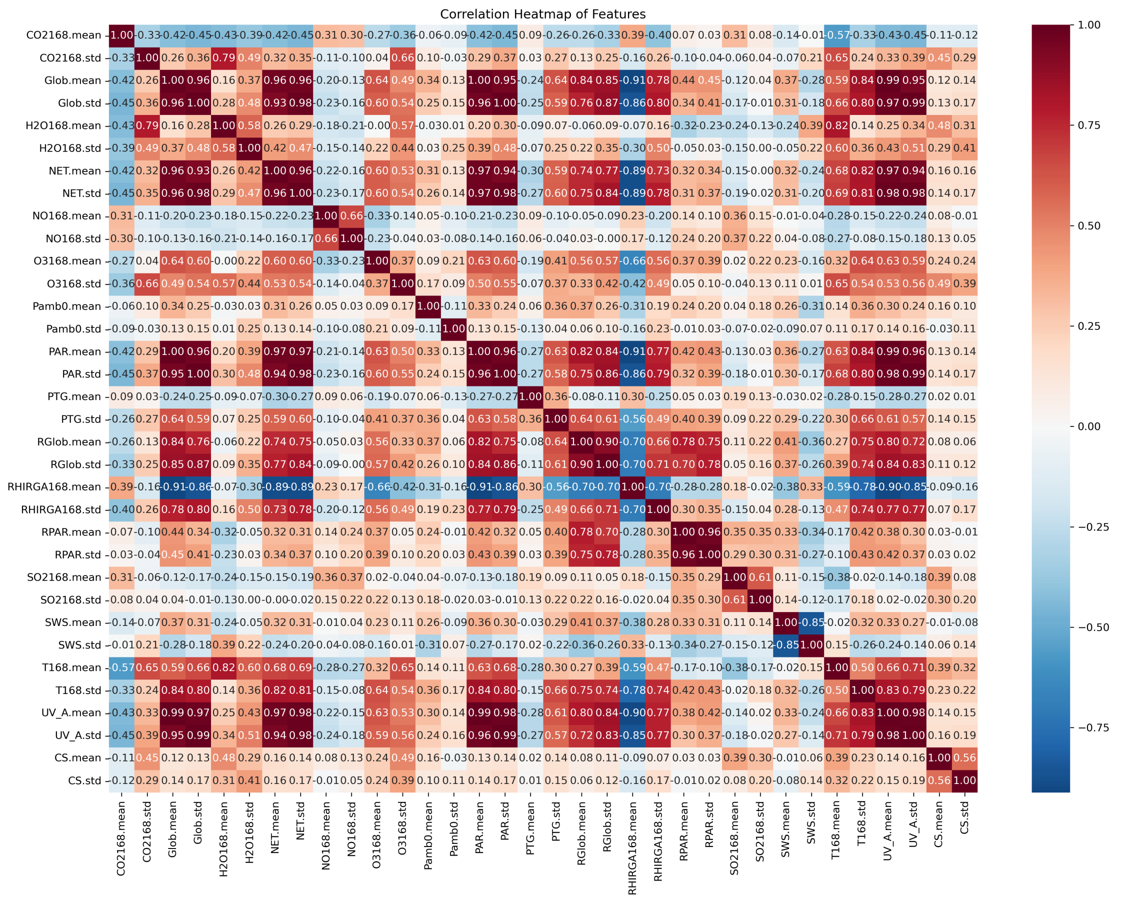

Figure 1: Heatmap of feature correlations for representative features. Strong positive correlations (red) are visible among radiation features, while negative correlations (blue) appear between radiation and humidity features.

2.5 Feature Distributions by Class

For this let's create a column class2 that is event if class4 is nonevent, and nonevent otherwise:

```
df_train["class2"] = df_train["class4"].apply(lambda x: "nonevent"
    if x == "nonevent" else "event")
```

Now let's look at the distribution of features by class2:

```
class2_means = df_train.groupby("class2")[example_features].mean()
class2_mean_diff = abs(class2_means.loc["event"] - class2_means.loc
    ["nonevent"])
class2_mean_diff_normalized = class2_mean_diff / df_train[
    example_features].std()
class2_top_features = class2_mean_diff_normalized.sort_values(
    ascending=False)

print("top features by normalized mean different between classes (
    class2):")
```

```
for feature, diff in class2_top_features.items():
    print(f"{feature}: {diff:.3f}")
```

```
top features by normalized mean different between classes (class2):
RHIRGA168.mean: 1.359
Glob.mean: 1.268
PAR.mean: 1.245
UV_A.mean: 1.186
NET.std: 1.160
Glob.std: 1.158
NET.mean: 1.148
RHIRGA168.std: 1.148
PAR.std: 1.137
RGlob.mean: 1.088
UV_A.std: 1.086
T168.std: 1.058
RGlob.std: 1.051
O3168.mean: 0.905
PTG.std: 0.814
SWS.mean: 0.628
CO2168.mean: 0.570
RPAR.mean: 0.570
RPAR.std: 0.559
CS.mean: 0.484
SWS.std: 0.484
PTG.mean: 0.432
H2O168.mean: 0.425
T168.mean: 0.375
Pamb0.std: 0.375
Pamb0.mean: 0.371
NO168.mean: 0.318
H2O168.std: 0.267
SO2168.mean: 0.260
O3168.std: 0.249
NO168.std: 0.220
CO2168.std: 0.179
SO2168.std: 0.062
CS.std: 0.048
```

Strong predictors:

- Relative humidity (RHIRGA168) and radiation features (Glob, PAR, UV_A, NET) show strong separation
- Temperature variability (T168.std) also show strong separation
- Event days appear to have lower humidity and higher radiation (sunny days)

Moderate predictors:

- Ozone and PTG variability show reasonable separation

- CO2 and reflected radiation (RPAR, RGlob) may contribute in combination with other features

Weak predictors:

- Condensation sink (CS.mean), temperature mean (T168.mean), pressure (Pamb0) and NO and SO2 show limited separation on their own
- CS.std has almost no separation despite being pretty relevant with weather?

The std features rank differently than the same feature means. This maybe suggests that stable vs fluctuating conditions matter more than the measurements themselves.

3 Data Preprocessing

3.1 Target Variable

I created a binary target variable class2 defined as *event* if class4 is Ia, Ib, or II, and *nonevent* otherwise.

3.2 Scaling

All features were standardized using StandardScaler for logistic regression models. Tree-based models (Random Forest) do not require scaling.

3.3 Feature Selection

I did not perform explicit feature selection for several reasons:

- Lasso regularization performs implicit feature selection by zeroing coefficients
- The high correlation among radiation features is handled by regularization

I considered PCA for dimensionality reduction but decided against it to maintain interpretability and because regularized logistic regression was already performing well.

4 Modeling

4.1 Validation Strategy

All models were evaluated using stratified 5-fold cross-validation to ensure reliable performance estimates despite the limited data. Five folds provide a reasonable bias-variance tradeoff With 450 training samples, each fold uses 360 samples for training and 90 for validation. I report mean accuracy and standard deviation across folds.

All models were evaluated using stratified 5-fold cross-validation to ensure reliable performance estimates despite the limited dataset of 450 samples. This validation approach provided a reasonable bias-variance tradeoff, with each fold using 360 samples for training and 90 for validation.

The cross-validation followed a consistent pattern throughout the project:

- Stratified Sampling: preserving class distributions in each fold
- Standardized Pipelines: feature scaling integrated with model training
- Consistent Scoring: accuracy as the primary evaluation metric

The core implementation pattern (shown in Appendix C) was adapted for each model variation by modifying classifier parameters (changing regularization type from L1 to L2, adjusting C values for hyperparameter tuning).

All performance metrics reported represent the mean and standard deviation across the five cross-validation folds.

4.2 Model Selection

I compared two main models:

Logistic Regression was chosen as the primary model because:

- Provides a strong baseline with good interpretability
- Handles redundant features through L1 (Lasso) or L2 (Ridge) regularization
- Works well with small datasets due to fewer parameters than complex models
- Produces well-calibrated probability estimates

Random Forest was tested as an alternative because:

- Handles feature redundancy through random feature subsampling
- Provides feature importance rankings
- Does not require feature scaling

I did not explore more complex models (XGBoost, neural networks) because logistic regression already performed well, and the small dataset size favored simpler models with fewer parameters to avoid overfitting.

5 Submissions and Results

5.1 Submission 1: Binary Classification Only

5.1.1 Approach

For the first submission, I focused only on binary classification (event vs. nonevent). Three models were compared using 5-fold cross-validation with the standard evaluation pattern:

- **Logistic Regression (Ridge)**: L2 regularization with $C = 1.0$
- **Logistic Regression (Lasso)**: L1 regularization with $C = 1.0$
- **Random Forest**: 100 estimators with default parameters

Cross-validation results showed Lasso regularization performed best:

```
Logistic Regression (Ridge): 0.860 (std 0.044)
Logistic Regression (Lasso): 0.884 (std 0.026)
Random Forest: 0.860 (std 0.030)
```

Lasso performed best, likely because of its implicit feature selection, zeroing coefficients for redundant radiation features. Random Forest underperformed, possibly due to overfitting on the small dataset.

5.1.2 Submission

Since the model only performs binary classification, I predicted *II* (the most common event type) for all days classified as events.

```
model.fit(df_x_train, df_y_train)
probs = model.predict_proba(df_x_test)[:, 1]
class4_pred = np.where(probs > 0.5, "II", "nonevent")
```

5.1.3 Result

Kaggle Score: 0.75712

Binary classification works well, but always predicting *II* for events hurts the multi-class accuracy component.

5.2 Submission 2: Direct Multi-class Classification

5.2.1 Approach

To improve the score, I attempted direct 4-class prediction using the same three model types. The same cross-validation pattern was applied to the multi-class problem.

Cross-validation accuracy decreased substantially compared to binary classification:

```
Logistic Regression (Ridge): 0.649 (std 0.023)
Logistic Regression (Lasso): 0.660 (std 0.029)
Random Forest: 0.656 (std 0.032)
```

As expected, multi-class is a lot harder than binary. Let's look more closely at class performance to see if we can do something.

Looking at the confusion matrices and classification reports we see that:

- Nonevent is reliable: 87-90% recall, 85% precision
- Ia remains the hardest class: Only 4-19% recall
- Ib and II are frequently confused: 34-46 Ib predicted as II across models
- Ridge has best Ia performance: 5 correct (19% recall) vs Random Forest 1 correct (4%)
- Lasso predicts II most aggressively: Highest II recall (59%) but lowest Ib recall (29%)
- Random Forest has lowest variance (std 0.008) but worst overall class balance

5.2.2 Submission

The accuracy was poor, but let's see what score we get with Logistic Regression with Ridge since it showed the best per-class balance.

```python
model.fit(df_x_train, df_y_train)
probs = model.predict_proba(df_x_test)
class4_pred = model.predict(df_x_test)
p_event = 1 - probs[:, list(model.classes_).index("nonevent")]
```

5.2.3 Result

Kaggle Score: 0.75238

Slightly worse than Submission 1, confirming that forcing the model to distinguish event subtypes hurts overall performance.

5.3 Submission 3: Two-Stage Classification

5.3.1 Approach

Based on the observation that binary classification works well but multi-class struggles, I implemented a two-stage approach:

1. **Stage 1:** Binary classifier (event vs. nonevent) using Logistic Regression with Lasso

2. **Stage 2:** Event subtype classifier (Ia, Ib, II) using Logistic Regression with Ridge, trained only on event samples

This decomposition allows each stage to specialize: Stage 1 learns the event/nonevent boundary from all data, while Stage 2 focuses on the more subtle distinctions between event types using only the 225 event samples.

5.3.2 Submission

```
# Stage 1
binary_model = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(penalty="l1", solver="saga", C=1.0))
])
binary_model.fit(df_x_train, df_y_train_binary)

# Stage 2
event_mask = df_train["class4"] != "nonevent"
event_model = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(penalty="l2", C=1.0))
])
event_model.fit(df_x_train[event_mask], df_y_train_events[
   event_mask])

# Combine predictions
p_event = binary_model.predict_proba(df_x_test)[:, 1]
binary_pred = (p_event > 0.5).astype(int)
event_pred = event_model.predict(df_x_test)

class4_pred = []
for i in range(len(df_x_test)):
    if binary_pred[i] == 0:
        class4_pred.append("nonevent")
    else:
        class4_pred.append(event_pred[i])
```

5.3.3 Result

Kaggle Score: 0.76921

A significant improvement! The two-stage approach successfully leverages the strong binary classification while adding meaningful multi-class predictions.

5.4 Submission 4: Hyperparameter Tuning

5.4.1 Approach

I attempted to optimize the regularization parameter C for both stages using grid search across values $[0.01, 0.1, 0.5, 1.0, 2.0, 10.0]$.

Stage 1:

```
stage 1
C=0.01: 0.831 (std 0.030)
C=0.1: 0.871 (std 0.023)
C=0.5: 0.893 (std 0.019)
C=1.0: 0.893 (std 0.042)
C=2.0: 0.887 (std 0.019)
C=10.0: 0.878 (std 0.038)
```

Stage 2:

```
stage 2
C=0.01: 0.547 (std 0.062)
C=0.1: 0.591 (std 0.033)
C=0.5: 0.560 (std 0.065)
C=1.0: 0.564 (std 0.033)
C=2.0: 0.538 (std 0.033)
C=10.0: 0.516 (std 0.033)
```

Based on cross-validation, I selected $C = 0.5$ for Stage 1 and $C = 0.1$ for Stage 2.

5.4.2 Result

Kaggle Score: 0.76162

Surprisingly, the tuned parameters performed worse than the defaults. This may be due to:

- Cross-validation variance with small sample sizes
- Overfitting to the cross-validation splits
- The default $C = 1.0$ providing a good balance between bias and variance

5.5 Submission 5: Probability Calibration

5.5.1 Approach

To optimize the perplexity component of the score, I applied sigmoid calibration to the binary classifier using CalibratedClassifierCV with 5-fold cross-validation.

5.5.2 Submission

```
base_model = LogisticRegression(penalty="l1", solver="saga", C=1.0,
    max_iter=5000)
binary_model = CalibratedClassifierCV(base_model, cv=5, method="
   sigmoid")
binary_model.fit(df_x_train_scaled, df_y_train_binary)

p_event = binary_model.predict_proba(df_x_test_scaled)[:, 1]
binary_pred = (p_event > 0.5).astype(int)
event_pred = event_model.predict(df_x_test)
```

5.5.3 Result

Kaggle Score: 0.76714

Calibration did not improve results. Logistic regression already produces well-calibrated probabilities, so additional calibration provided no benefit and may have introduced additional variance.

6 Discussion

6.1 What Worked

Two-stage classification was the key insight that improved performance. By decomposing the problem into binary classification followed by event subtype classification, each stage could specialize on its task without interference.

Lasso regularization effectively handled the redundant radiation features, performing implicit feature selection. The L1 penalty zeroed coefficients for correlated features, reducing overfitting.

Simple logistic regression outperformed Random Forest, likely because the small dataset (450 samples) favored simpler models with fewer parameters. Complex models with many hyperparameters tend to overfit with limited data.

6.2 What Didn't Work

Hyperparameter tuning did not improve results, possibly because:

- Cross-validation estimates are noisy with small samples
- Default parameters were already near-optimal for this problem
- Risk of overfitting to the validation folds

Probability calibration was unnecessary since logistic regression already produces calibrated probabilities.

Random Forest underperformed expectations, likely overfitting despite the bagging procedure. With only 450 samples, each tree saw limited data.

6.3 Reflection on Performance

My best score of 0.76921 placed 13th on the leaderboard. The main bottleneck remains multi-class classification, distinguishing between Ia, Ib, and II event types is inherently difficult with only 225 event samples (26 Ia, 82 Ib, 117 II).

The score was approximately what I expected based on cross-validation estimates. Binary classification achieved around 88% accuracy in CV, while event subtype classification achieved only around 56%. The combined Kaggle score reflects this imbalance.

6.4 Lecture Insights

After attending the project lecture where top performers presented their approaches, several key observations emerged:

- Most of the top teams used a heuristic tree approach, similar to my two-stage approach, confirming this as an effective approach. The second-place team extended it further to a heuristic tree of binary classifiers.
- Hyperparameter tuning and feature engineering helped very little, typically giving only 0.005-0.01 score improvements across teams.
- The one team achieved good scores with science-informed feature engineering, not algorithmic complexity.

These confirm that my methodology was fundamentally good and aligned with successful approaches. The main differentiator for top positions was domain knowledge and a more developed approach.

6.5 Potential Improvements

For future work, I would consider:

- Domain-informed feature engineering (interaction terms based on physical relationships)
- Extending to a heuristic tree of binary classifiers
- Ensemble methods for robustness
- Cost-sensitive learning for the minority Ia class

7 Self-Grading

Proposed Grade: 4

7.1 Justification

I believe my work merits a grade of 4 (very good) based on the following assessment:

Strengths:

- Thorough data exploration with clear explanations of feature characteristics, correlations, and class distributions
- Systematic model comparison using appropriate cross-validation methodology
- Clear documentation of the iterative process, including both successful and unsuccessful attempts
- The two-stage classification approach demonstrates independent problem-solving and creative thinking
- Appropriate use of regularization to handle feature redundancy
- Results achieved (0.769) are competitive, placing 13th on the leaderboard

Areas for improvement:

- Limited exploration of feature engineering or domain-specific features
- Could have included more visualizations
- Could have explored more model families or ensemble methods

I think my work shows solid understanding of machine learning methods, appropriate model selection for the problem characteristics (small dataset, redundant features), and honest reflection on what worked and what didn't. The reporting is clear and follows a logical structure, though more visualizations would enhance readability. The work was completed within the scheduled timeline.

A Selected Features for Analysis

The following features were selected as representatives from each "group" for data exploration:

```
CO2168.mean    # CO2
CO2168.std     # CO2
Glob.mean      # Global radiation
Glob.std       # Global radiation
H2O168.mean    # Humidity
H2O168.std     # Humidity
NET.mean       # Net radiation
NET.std        # Net radiation
NO168.mean     # NO concentration
NO168.std      # NO concentration
O3168.mean     # O3 concentration
```

```
O3168.std       # O3 concentration
Pamb0.mean      # Ambient pressure
Pamb0.std       # Ambient pressure
PAR.mean        # Photosynthetically active radiation
PAR.std         # Photosynthetically active radiation
PTG.mean        # Pressure tendency gradient
PTG.std         # Pressure tendency gradient
RGlob.mean      # Reflected global radiation
RGlob.std       # Reflected global radiation
RHIRGA168.mean  # Relative humidity
RHIRGA168.std   # Relative humidity
RPAR.mean       # Reflected PAR
RPAR.std        # Reflected PAR
SO2168.mean     # SO2 concentration
SO2168.std      # SO2 concentration
SWS.mean        # Soil water storage
SWS.std         # Soil water storage
T168.mean       # Temperature
T168.std        # Temperature
UV_A.mean       # UV-A radiation
UV_A.std        # UV-A radiation
UV_B.mean       # UV-B radiation
UV_B.std        # UV-B radiation
CS.mean         # Condensation sink
CS.std          # Condensation sink
```

B Full code for finding high correlation pairs

```
example_features_mean = [feature for feature in example_features if
    ".mean" in feature]
example_features_std = [feature for feature in example_features if
   ".std" in feature]
correlation_matrix_mean = df_train[example_features_mean].corr()
correlation_matrix_std = df_train[example_features_std].corr()

high_correlation_pairs = []
for i in range(len(correlation_matrix_mean.columns)):
    for j in range(i + 1, len(correlation_matrix_mean.columns)):
        if abs(correlation_matrix_mean.iloc[i, j]) > 0.9:
            high_correlation_pairs.append(
                {
                    "feature1": correlation_matrix_mean.columns[i],
                    "feature2": correlation_matrix_mean.columns[j],
                    "correlation": correlation_matrix_mean.iloc[i,
                       j],
                }
            )
for i in range(len(correlation_matrix_std.columns)):
    for j in range(i + 1, len(correlation_matrix_std.columns)):
        if abs(correlation_matrix_std.iloc[i, j]) > 0.9:
            high_correlation_pairs.append(
                {
                    "feature1": correlation_matrix_std.columns[i],
                    "feature2": correlation_matrix_std.columns[j],
                    "correlation": correlation_matrix_std.iloc[i, j
                       ],
                }
            )
```

C Code for cross-validation

```
cv = StratifiedKFold(n_splits=5, shuffle=True)

# Example pipeline for Logistic Regression with Lasso
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(
        penalty="l1",
        solver="saga",
        C=1.0,
        max_iter=5000
    ))
])

scores = cross_val_score(
    pipeline,
    X_train,
    y_train,
    cv=cv,
    scoring="accuracy"
)
print (f"Logistic Regression (Lasso): {scores.mean():.3f} (std {
   scores.std():.3f})")
```