



**UNIVERSITÀ  
DI TORINO**

# Machine Learning for Photometric Supernova Classification

Machine Learning for Applied Physics and High Energy Physics

**Riccardo Zangarini**

Major in Astrophysics and Theoretical Physics  
University of Turin

January 23, 2024



# Overview

1. Dataset
2. Feature Extraction
3. PCA
4. Clustering
5. Supervised Learning
6. Conclusions



# Dataset

---

# The Supernova Classification Challenge

Blinded mix of simulated SNs, with types (Ia, Ib, Ic, II) selected in proportion to their expected rate. The simulation is realized in the *griz* filters of the Dark Energy Survey (DES) with realistic observing conditions.

Motivations:

- Huge, wide-field surveys that provide access to an ever-increasing amount of data,
- SNs are of fundamental importance for cosmological studies, in particular for constraining the parameters of the  $\Lambda - CDM$  model,
- Small portion of SN observations can be identified spectroscopically,
- Wide availability of light curves (photometric measurements) .

# The Supernova Classification Challenge

Training not representative in brightness or redshift  $z$ . This aims to emulate the way spectroscopic measures are currently performed, prioritizing low-redshift, bright objects.

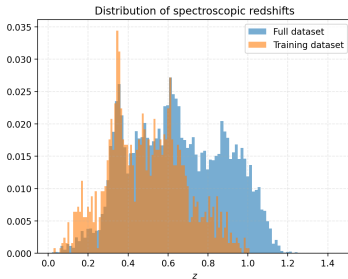


Figure: 1: Relative frequency of SN observations in terms of redshift  $z$ .

# Preprocessing

We only have 1250 labelled supernova events available.

First steps:

- Clean the dataset by removing 'nan' values.
- Plot the Light curves to make sure everything is all right.

## What is a SN Light Curve?

Graphical representation of the changes in brightness (luminosity) of the supernova over time. It shows how the luminosity of the supernova evolves from the moment it is first observed until it fades away.

# Light Curve

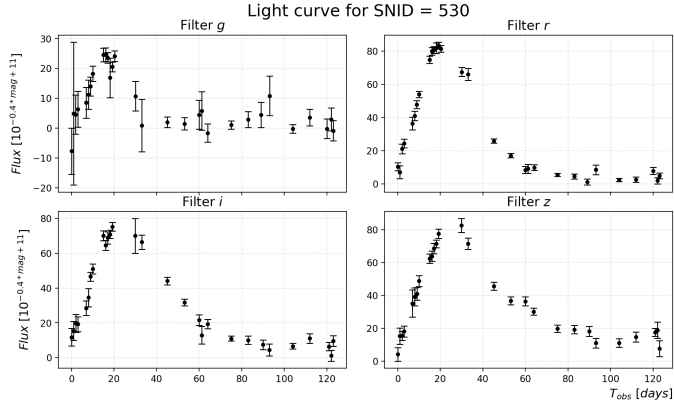


Figure: 2: Light Curves for each of the *griz* filters. The Flux (y-axis) is a function of the magnitude  $m$ .



# Feature Extraction

---



# Feature Extraction

The light curves are characterised by multiple points per filter, resulting in a very large number ( $\gtrsim 100$ ) of possible features. A first step of dimensionality reduction is necessary: **feature extraction**.

A 6-parameter function is used to parameterise each curve, resulting in 24 parameters for each SN:

$$f(t) = A \left[ 1 + B(t - t_1)^2 \right] \cdot \left( \frac{e^{-(t-t_0)/T_f}}{1 + e^{-(t-t_0)/T_r}} \right) \quad (1)$$

Such a function is taken from *Karpenka et al., 2013*.

- No actual physical meaning but it is sufficiently general to fit the shape of almost any SN light curve.
- No offset.

# Feature Extraction

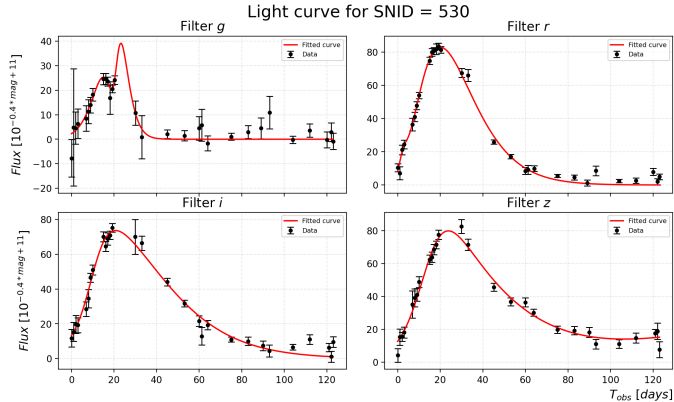


Figure: 3: Fitted Light Curves for each of the *griz* filters. Equation (1) is able to intercept double peaks.

# Meaningless Features

Exploit the *glimpse* we get from observing parameter distributions to determine which of them can be discarded or modified.

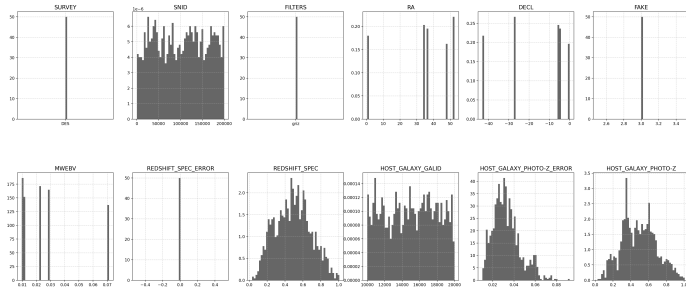


Figure: 4: Histograms for all the non-extracted features. We can discard 4 of them.

# Redundant Features

We use the **correlation matrix**, a table that shows the correlation coefficients (Pearson coefficients  $r$ ) between many variables.

- $r = (-)1$ : perfect linear (anti)correlation,
- $r = 0$ : no linear correlation.

Concerning the Light Curve parameters:

- Same parameter in different filters shows slight positive correlation,
- $T_f$  and  $t_1$  present correlation up to 0.68. No removal at the end.

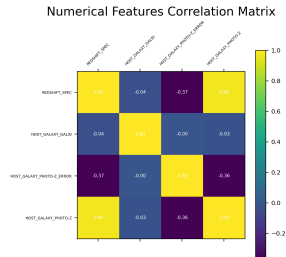


Figure: 5: High  $z$  correlation.



# PCA

---

University of Turin

# Elbow Point

Identify the optimal number of components to be retained during PCA by locating the **elbow point** in the eigenvalues of the covariance matrix: at the elbow point, the eigenvalues start to plateau, suggesting diminishing returns in terms of variance explained.

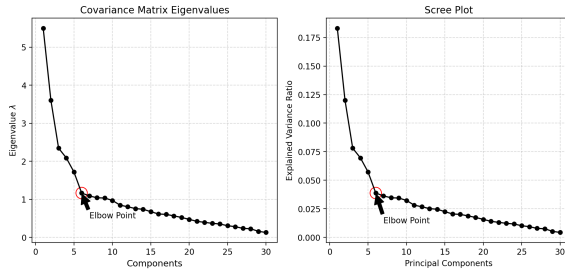


Figure: 6: We can identify the elbow point at the 6<sup>th</sup> Principal Component (PC).

# Cumulative Variance

We can also fix the cumulative explained variance value and derive the number of principal components needed to achieve this value.

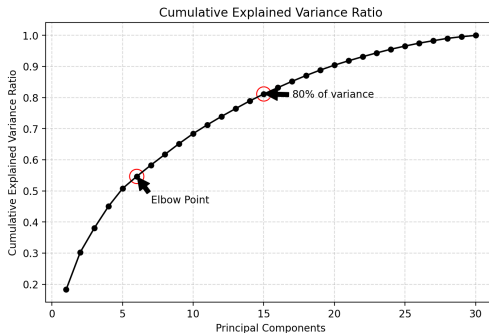


Figure: 7: The number of features at the elbow point preserves only about 56% of variance.

# Features and PCs Relation

To find the relations between the Principal Components (PCs) and the original features, we examine the **loading matrix**, denoted as  $\mathcal{L}$ : provides the coefficients that express the linear combination of the original features to obtain each principal component.

$$PC_i = \mathcal{L}_i^j X_j \quad (2)$$

where:

- $PC_i$  is the  $i$ -th Principal Component,
- $\mathcal{L}_i^j$  is the loading coefficient of the  $j$ -th original feature in the  $i$ -th Principal Component,
- $X_j$  is the  $j$ -th original feature.



# Features and PCs Relation

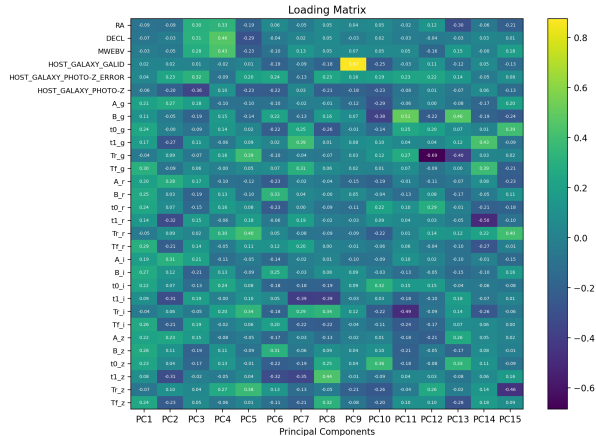


Figure 7: Loading matrix after PCA fitting.



# Clustering

---

# KMeans

Two-step Algorithm: **Instance labeling** assigns each instance to the closest centroid and **Centroid Update** computes the mean of the instances that belong to the same cluster updating the centroid position.

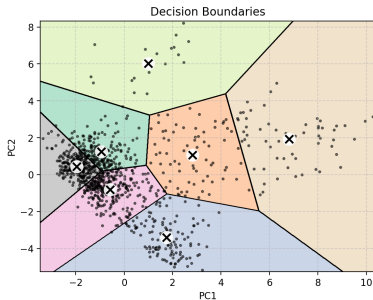


Figure: 8: KMeans clustering on the dataset projected along the first 2 PCs.

# KMeans

## Problems of KMeans

KMeans is a partitioning clustering algorithm that works well when clusters are well-separated, roughly spherical, and of similar sizes. The 2D projection of our PC-projected dataset does not seem to meet any of these demands.

Develop the same method on the  $d$ -dimensional PC-projected space: we know the true labels and we can use them to assess the model's accuracy. We get a very low Rand Index:  $RI = 0.477$  and inadequate Purity  $p = 0.702$ .

- Poor separability, superposition, non-spherical shapes.
- Highly imbalanced dataset.

# Confusion Matrix

Most points are misclassified and the most populated class ends to be lbc.

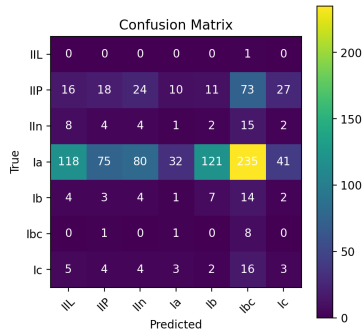


Figure: 9: Confusion matrix for the KMeans model trained on the first 15 PCs.

# Gaussian Mixture Models

Probabilistic models that assume that the instances were generated from sampling from a mixture of several Gaussian distributions whose parameters are unknown. The dataset  $\mathbf{X}$  is assumed to have been generated from the following probabilistic process:

- For each instance, pick a random cluster among  $k$  clusters. Define *cluster weight*  $= \Phi^{(j)}$ , that is the probability of choosing the  $j^{th}$  cluster. The index of the cluster chosen for the  $i^{th}$  instances is denoted as  $z^{(i)}$ . So, if the  $i^{th}$  instance was assigned to the  $j^{th}$  cluster, we can write  $z^{(i)} = j$ .
- Instance location sampled from a Gaussian distribution:  $\mathbf{x}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)})$

# Final Model

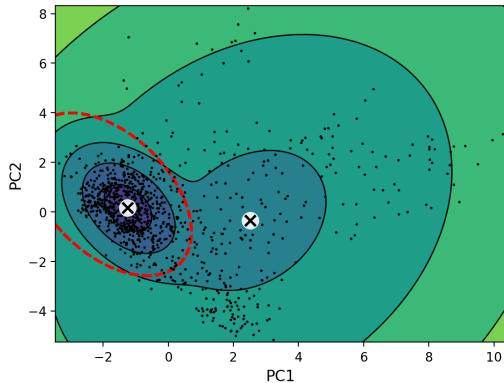


Figure: 10: Decision Boundary for the GMM model trained on the first 2 PCs.

# Final Model

”Better” confusion matrix, at least collecting most of the instances into the I class. Now we want to implement a classifier to evaluate the effects of unsupervised pre-processing on the available data. In particular, we will study the effects of **dimensionality reduction** obtained by feature-extraction and PCA, and evaluate whether the additional information produced by **clustering** will help the classification algorithm.

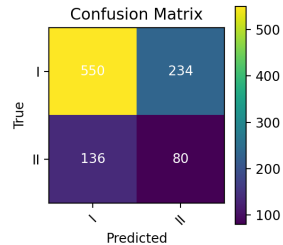


Figure: 11: GMM confusion matrix.





# Supervised Learning

---

# Data Augmentation - SMOTE

The dataset is severely **imbalanced**. Plus, we can not implement under-sampling.

- *Identify Minority Class Instances*: SMOTE is applied to each minority class.
- *Select a Minority Instance*: randomly choose an instance from the minority class.
- *Find Nearest Neighbors*: identify the  $k$ -nearest neighbors of the selected instance.
- *Generate Synthetic Instances*: for each nearest neighbor, create synthetic instances along the line connecting the selected instance and its neighbor.
- *Repeat*: repeat the process until balance is achieved.

# Dataset

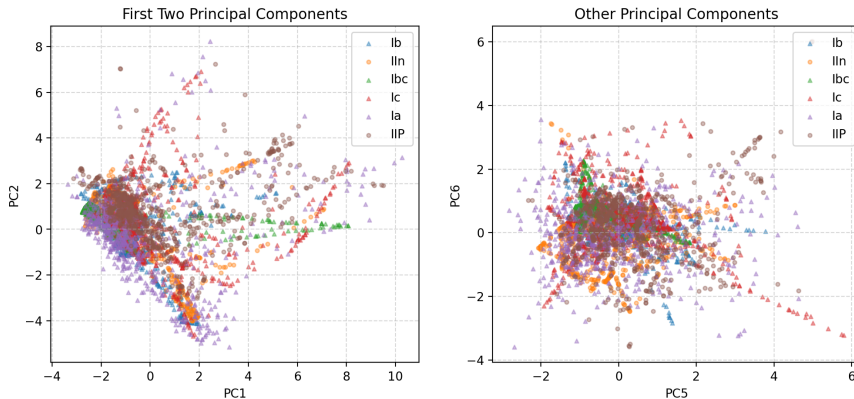


Figure: 11: Augmented dataset projected on the first 2 PCs (right) and on PC5 and PC6 (left).

# Neural Network

- **Fully-Connected:** each node in one layer is connected to every node in the next layer. The connections have associated weights that the network learns during the training process.
- **Feedforward:** information flows in one direction, from the input layer through the hidden layers to the output layer. There are no cycles or loops. This is true also for gradient computation using *backpropagation*.
- **Training Set:** 1250 PC-projected data points, each consisting in 15 PCA features plus 1 feature extracted with the GM model.

# Neural Network

For each layer: **L2 Regularization**,  $\lambda = 0.005$ .

Between each layer: **Batch Normalization** and **Dropout**.

Number of trainable parameters:  $n_p^l = (n_{in}^l + 1) \cdot n_{out}^l$ .

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 300)	5100
dense_1 (Dense)	(None, 300)	90300
dense_2 (Dense)	(None, 100)	30100
dense_3 (Dense)	(None, 6)	606
Total params: 126106 (492.60 KB)		
Trainable params: 126106 (492.60 KB)		
Non-trainable params: 0 (0.00 Byte)		

Figure: 12: Model Architecture.

# Training

- `model.compile(loss='sparse_categorical_crossentropy',  
optimizer=keras.optimizers.Adam(learning_rate=0.005),  
metrics=['accuracy'])`
- `model.fit(X_train, y_train, epochs=100, steps_per_epoch=64,  
validation_data=(X_val, y_val),  
callbacks=[keras.callbacks.EarlyStopping(patience=20,  
restore_best_weights=True),  
precision_recall_callback])`

The `precision_recall_callback` is a custom callback that computes *precision* and *recall* at the end of each epoch.

# Training

On a perfectly balanced dataset such this one, the **accuracy** is a fine metric. In fact, notice that it almost perfectly overlaps with *precision* and *recall*.

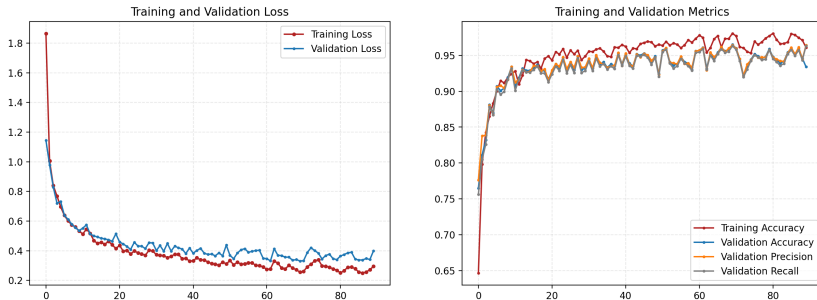


Figure: 12: Learning curves. Model trained on the complete dataset.

# Results

Table: 1: Results from some of the variations applied on the dataset. Top rows: *validation* results, bottom rows: *test* results.

Dataset	Loss Function	Accuracy	Precision - Recall	Time [s]
16- <i>d</i> , 6 labels	0.41	0.94	0.96 - 0.96	42.52
	1.69	0.66	0.31 - 0.29	//
15- <i>d</i> , 6 labels	0.38	0.96	0.96 - 0.96	43.16
	1.70	0.69	0.32 - 0.35	//
7- <i>d</i> , 6 labels	0.69	0.87	0.88 - 0.86	47.32
	2.03	0.53	0.29 - 0.31	//
6- <i>d</i> , 6 labels	0.64	0.87	0.87 - 0.87	43.53
	2.07	0.53	0.28 - 0.34	//



# Results

## Large Generalization Gap

Accuracy, precision and recall from the *validation* set are much higher than those resulting from the *test* set. **Why?**

Probably due to the strong imbalance between the classes also in the test set, where we have only 250 instances, with over 170 Type Ia SNs.

Perform **data augmentation** also on the test set? → NOT recommended (we may introduce biases that compromise the integrity of the evaluation).

One of the main risks is **Data Leakage**: introduction of patterns or information from the training set.

# Variation

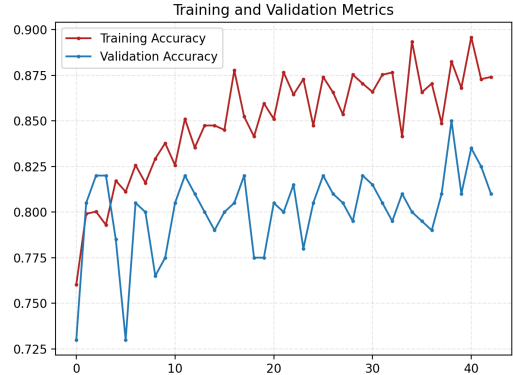
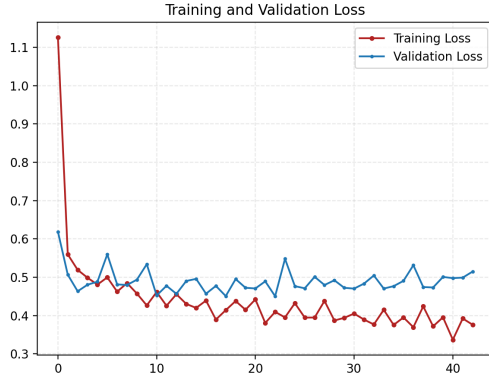


Figure: 13: Learning curves. Model trained on the non-augmented, binary dataset (Type Ia VS "Others").

# Results

Dataset	Loss Function	Accuracy	Precision - Recall	Time [s]
16-d, 2 labels	0.39	0.81	0.77 - 0.78	11.72
	0.51	0.79	0.75 - 0.74	//

Table: 2: Results using the non-augmented, binary dataset.

The test set is now much more **representative** of the training set. Despite lower accuracy, the model is more reliable and generalizes better.

## Underfitting

The accuracy is now lower due to the limited sample size of the training set.



# Conclusions

---

# Conclusions






**Problem:** very unbalanced dataset, reduced in size, with too many features (light curves). We applied:

- Feature Extraction: extract parameters from LCs successfully,
- PCA: greatly reduce dimensionality,
- Clustering: provide an extra feature that was very meaningful,
- Data Augmentation: more data points to combat underfitting.

**Possible improvements:** larger and less unbalanced dataset (observational problem), extract more meaningful features, data augmentation with numerical simulations.

This strategy proved to be a good starting point for analysing the many photometric observations already available and still to come.

# References

-  Karpenka et al. (2013)  
A simple and robust method for automated photometric classification of supernovae using neural networks
-  Kessler et al. (2010)  
Supernova Photometric Classification Challenge
-  Kessler et al. (2010)  
Results from the Supernova Photometric Classification Challenge
-  Mehta et al. (2019)  
A high-bias, low-variance introduction to Machine Learning for physicists
-  Géron (2023)  
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow



**UNIVERSITÀ  
DI TORINO**

# Thank you for your attention

**Riccardo Zangarini**

Major in Astrophysics and Theoretical Physics  
University of Turin

January 23, 2024