ATLAS Calorimeter Cluster Splitting

Guillaume Genthial

Institute for Computational and Mathematical Engineering

Problem

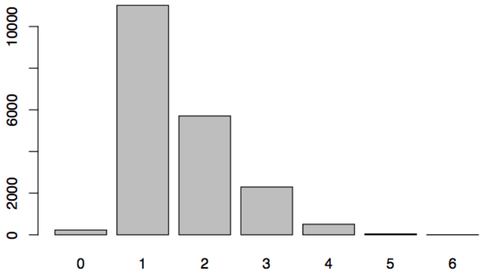


Figure: Nb of particles in a topo cluster

- Baseline: always predict 1 particle cluster
- Baseline performance: $\approx$ **56%** of accuracy

Formulation

- ideally, end-to-end clustering algorithm
- or given a topo cluster, decide how to split it
- simplify: predict the number of particles inside a topo-cluster (classification problem) $\rightarrow$ model that outputs probabiliy of each nb of particle
- 3 methods that only use calorimeter cells (no tracks)
  - **Simple features**: extract global features from the cluster + multi-layer perceptron (MLP)
  - **Multi-view**: create 2D images from the cells of the calorimeter + convolutions
  - **Embeddings**: get a meaningful representation of each cell + rotation-invariant operations to classify the cluster

Simple Features

Extract global features from the cluster

- total number of cells in the cluster
- energy, transverse momentum
- basic topology of the cluster (range in η and ϕ, depth, etc.)
- *shape* features:
  - take the 5 top cells (with highest energy deposition)
  - compute ΔR to the center of the cluster
  - add e_{cell}, pT_{cell} and Δ_{cell} to the features

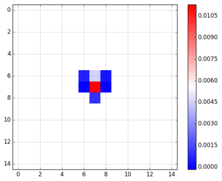
Feed each 24-dimensional feature vector into a MLP (100, 20)

Multi-view

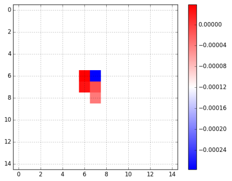
For each layer of the detector

- discretize the (η, ϕ) space. Resolution: $(0.1, 0.1)$, range $(1.5, 1.5)$
- extract features from each cell: energy, pT, energy density, volume...

A cluster $\rightarrow 15 \times 15 \times (24 \times \textit{nfeatures})$ image (channels)



(a) E Density - layer 6



(b) E Density - layer 7

Feed each image into a 2d-conv (3×3 filter, 100 channels) + MLP(1000)

Embeddings

We need to build a model that takes a list of cells and

- does not depend on the order of the cells
- takes the interaction between points into account (local structure)

Solution (PointNet, 3D Computer Vision): max-pool layer

- extract features from each cell: $(\eta, \phi, \mathit{depth}, \mathit{energy})$
- get an embedding for each cell $n \times 64$
- feed each embedding into a MLP $n \times 1024$
- perform a max-pooling on each component: get a vector $g = 1024$
- use g for prediction, or concatenate g to each embedding $n \times 1088$ and repeat

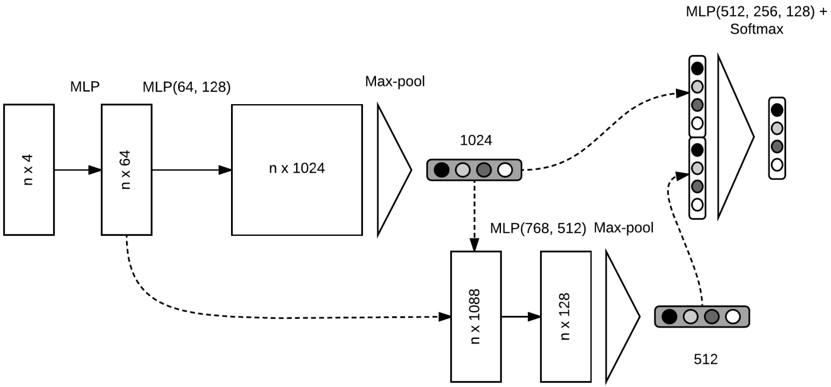


Figure: Embeddings architecture

Results

Data (ROOT) 2k events dataset (600k clusters).
Due to limited GPU resources, we trained our models on a fraction of the dataset (10%).
Implementation: tensorflow, Adam optimizer, lr=0.001

	$c = 2$	$c = 3$	Macro F1 $c = 3$
Baseline	56.71	56.71	24.13
MLP Simple Features	74.29	**62.75**	41.39
Conv(3×3) + MLP	74.33	62.28	**46.79**
Embeddings (proto)	**74.36**	62.20	42.37

Table: Classification Accuracy

Performance is for indicative purpose only (one run).

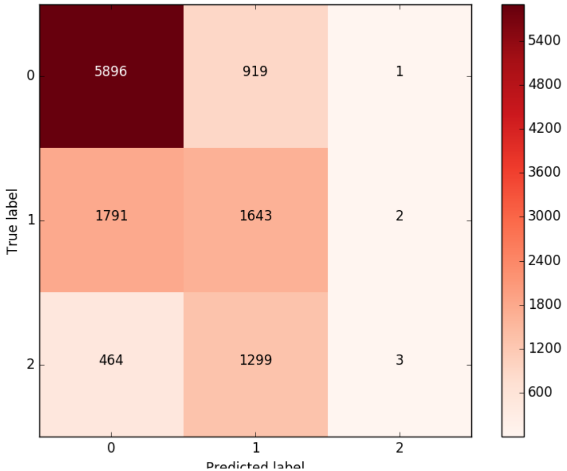


Figure: Confusion Matrix - Simple Features

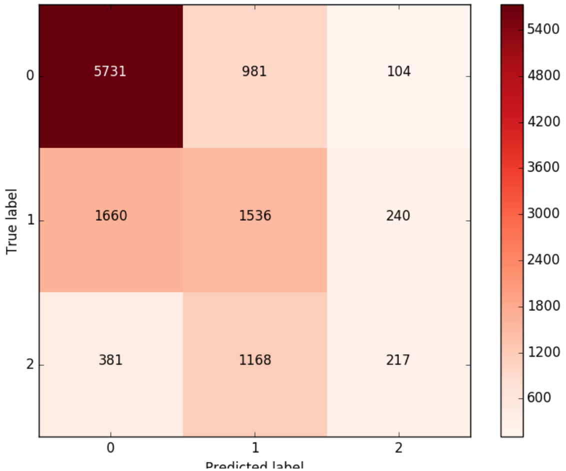


Figure: Confusion Matrix - Multi-view

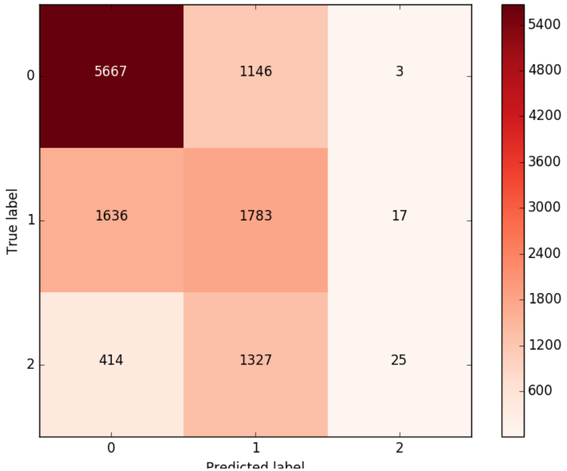


Figure: Confusion Matrix - Embeddings

Discussion

Model	Pros	Cons
Simple Features	simple, light	global, feature engineering
Multi-view	Local	sparsity, resolution
Embeddings	Local, dense, all cells	heavier

The **simple features** method performs really well. However, it only provides a global understanding (at the cluster level).
If we have a more complicated objective (new clusters position etc.), the **multi-view** and **embedding methods** are more promising, as they already capture local understanding (at the cell level).
The **embedding method** fixes the issue of sparsity and resolution that we had for the multi-view method and would benefit from more data + more precise objective.