A Peek Into ML for Collider Physics

Life could have been much easier if this tutorial were put into a Jupyter Notebook.

Yonghao Liu

Outline

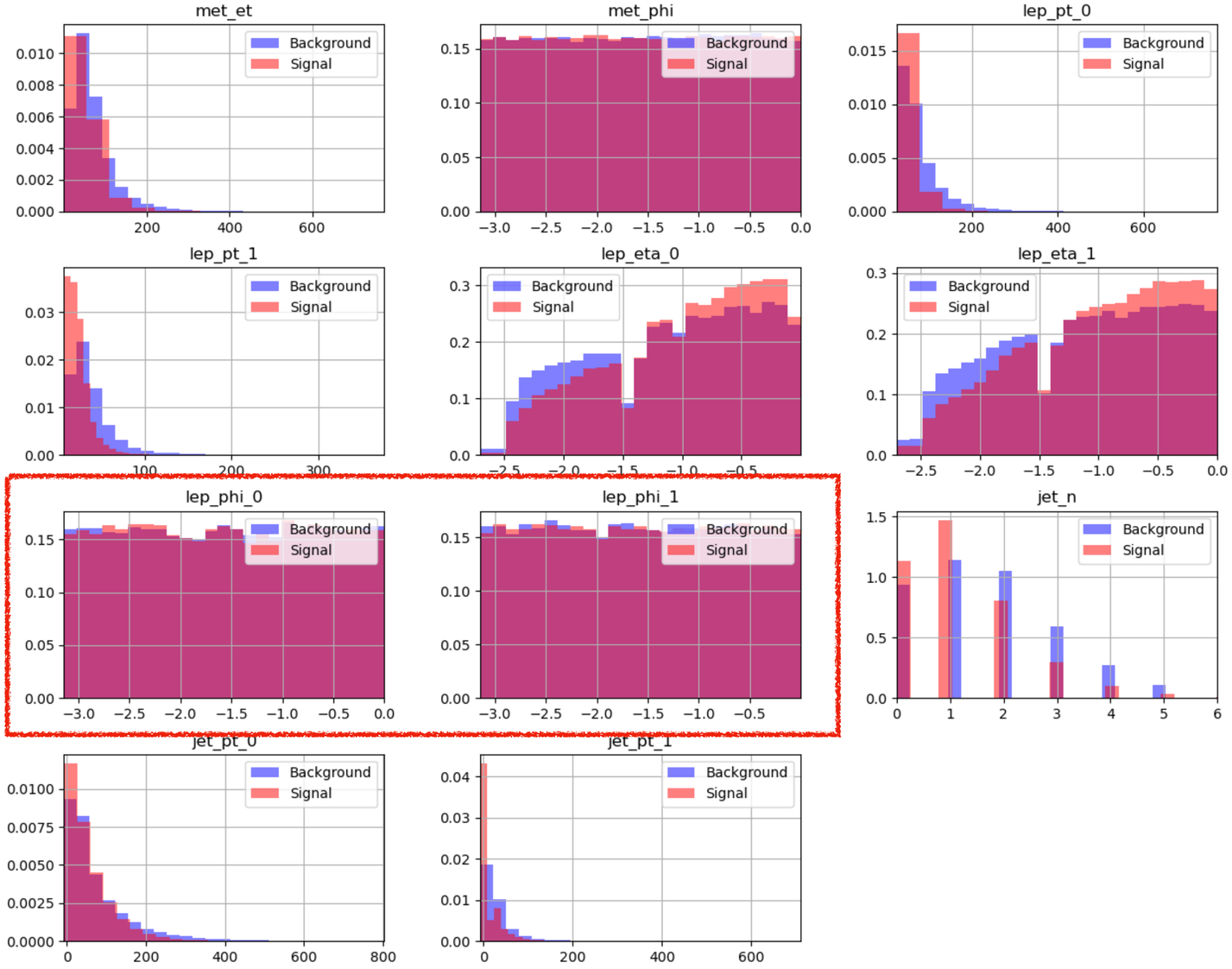
We learned from this tutorial…

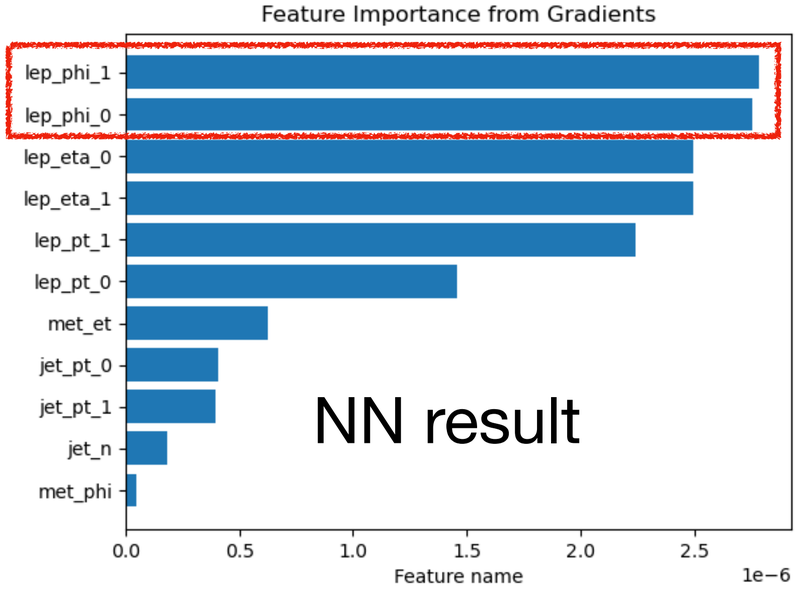
- How ML could be used in Collider Physics
- NN
- BDT (Not going to elaborate)
  - Combine many not so powerful things
  - Advantages: simple, interpretable

How ML is meant to be used in Collider Physics

- Simulation creates events with known labels (signal or background)
- ML learns from simulation how to categorize events
- Models learned to be used on real data
- Meant to replace traditional “cutting” method

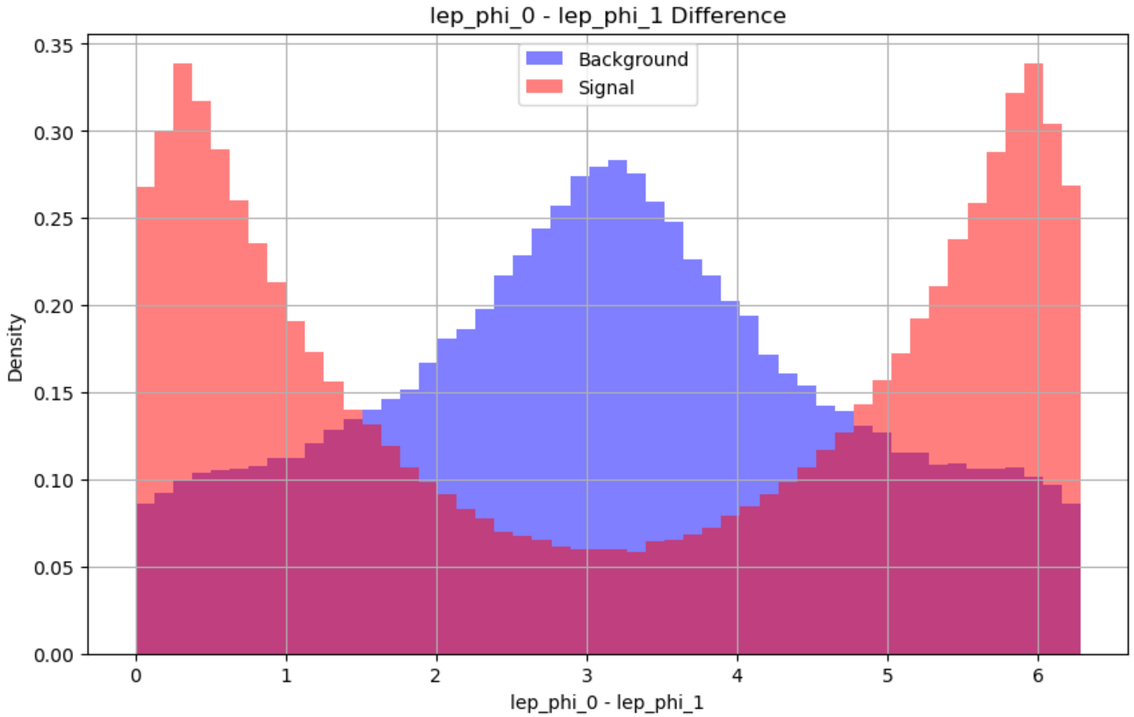
Feature importance





Feature importance

- $\Delta\phi_{lep}$ actually makes a difference!
- $\mu_{bkg} \approx \pi;\ \mu_{sig} \approx 0$
- 2 leptons from the background events are more likely to be derived from the same decay event of a neutral particle.

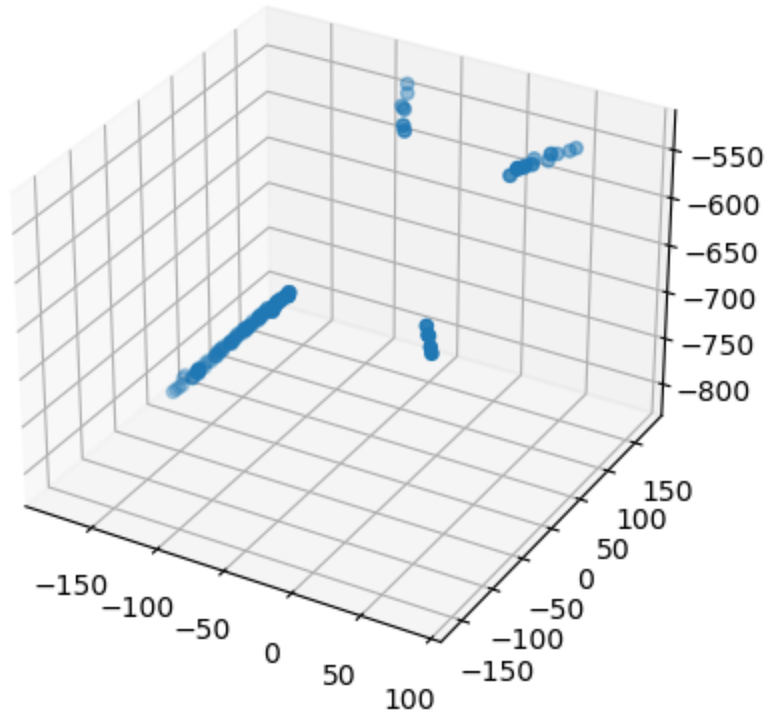


Lesson learned

- Derived features tell more than the raw features.

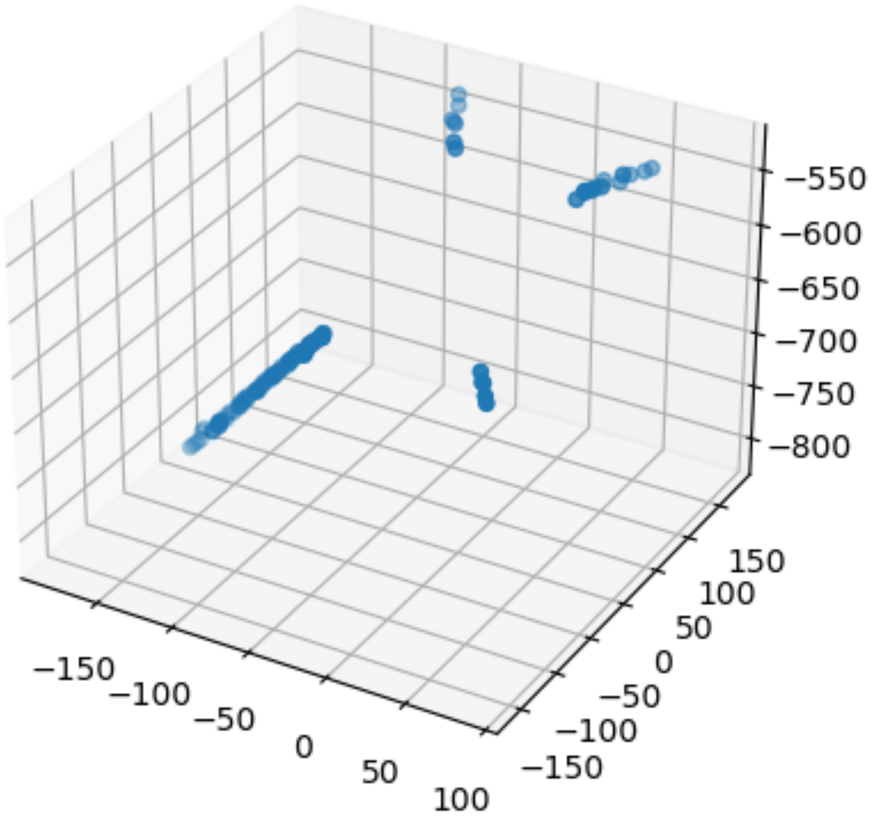
Sorry for bumping in , this is Alpha G

- Can we do the estimation without machine learning?

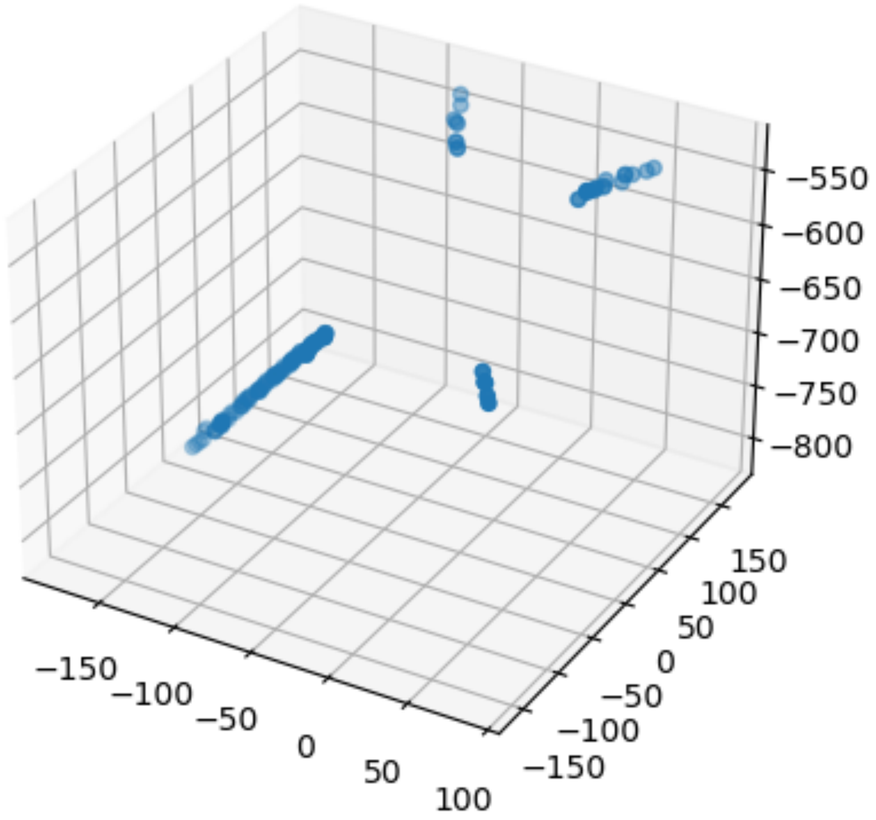


Grouping track

- Make an empty set for vectors
- Calculate the mean Z of the event
- Point cloud – mean Z
- Compare with the vectors in the set
- If Cosine >0.99, they are from the same track; if not, register the track in the set



Code

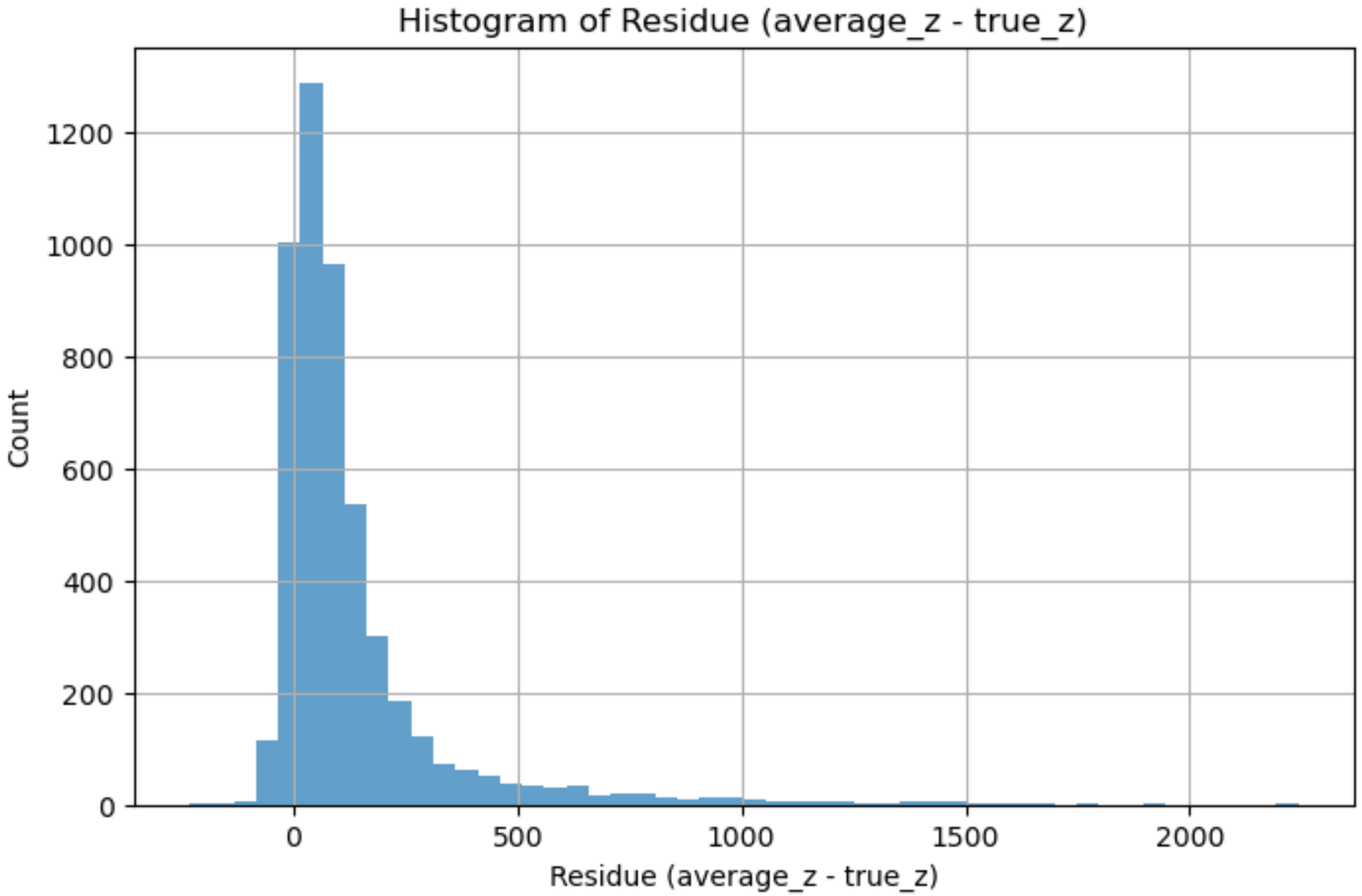


Group tracks

```python
import torch

def cosine_angle(a: torch.Tensor, b: torch.Tensor) -> float:

    a_norm = a / a.norm()
    b_norm = b / b.norm()
    return torch.dot(a_norm, b_norm).item()


def group_tracks(point_cloud: torch.Tensor, angle_threshold: float = 0.99):
    representatives = []
    group_labels = []
    center_of_mass = torch.tensor([0, 0, point_cloud[2].mean()])
    for idx in range(point_cloud.shape[1]): # 140
        vec = torch.tensor([point_cloud[0][idx], point_cloud[1][idx], point_cloud[2][idx]]) -
center_of_mass
        if vec.norm() < 1e-6:  # Skip zero vectors
            continue
        assigned = False
        for rep_idx, rep in enumerate(representatives):
            if cosine_angle(vec, rep-center_of_mass) > angle_threshold:
                group_labels.append(rep_idx)
                assigned = True
                break
        if not assigned:
            representatives.append(vec+center_of_mass)
            group_labels.append(len(representatives) - 1)
    return representatives, group_labels

representatives, group_labels = group_tracks(point_cloud)

print("Number of groups (tracks):", len(representatives))
print("Group labels for each vector:", group_labels)

print(representatives)
```

```
Number of groups (tracks): 4
Group labels for each vector: [0, 0, 0, 1, 0, 2, 0, 2, 0, 2, 0, 0, 1, 0, 0, 1, 0, 1, 0, 2, 2, 0, 1, 0, 2, 0, 0, 0, 2, 0, 1, 0, 1, 3, 3, 2, 1, 3, 3, 0, 0, 0, 1, 0, 2, 0, 2, 0, 2, 0, 0,
[tensor([  -20.5582,   115.7729, -1090.0000]), tensor([   53.1304,  -109.6045, -1090.0000]), tensor([ -64.7778,  106.3383, -958.0000]), tensor([  -31.2930,   168.0747, -1126.0000])]
```

Result


Histogram of Residue (average_z - true_z)
Count
0
200
400
600
800
1000
1200
0
500
1000
1500
2000
Residue (average_z - true_z)