Jet charge classification with ML

Meisam Ghasemi Bostanabad

Analysis meeting
2024-07-18

IPM

Introduction to Jet charge

1. What is Jet Charge?

- Jet charge is a measure of the electric charge of jets produced in high-energy particle collisions.
- It helps distinguish between particles and antiparticles, such as up quarks (u) and anti-up quarks (ū).

2. Importance of Jet Charge:

- Crucial for studying the properties of quarks and their interactions.
- Helps in identifying the type of quark that initiated the jet.

3. Challenges in Jet Charge Computation:

- High-dimensional and noisy data from particle detectors.
- Need for precise algorithms to correctly identify the charge.

4. Role of Machine Learning:

- Machine learning algorithms can analyze complex datasets and improve charge discrimination.
- Capable of learning patterns and making accurate predictions.

ML steps for jet charge computation

1. Event generation:

- 2M u+g and ū+g events using MG5 and Pythia8 in a notebook
- Pythia slowjet to construct small R-jets (0.2) and min pT and max Eta

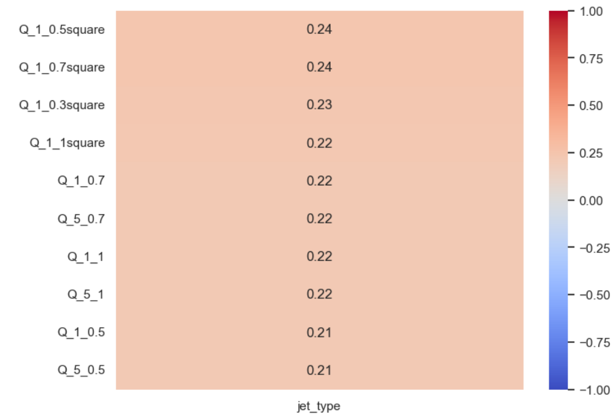
2. Feature Engineering:

- Key features for jet charge computation:

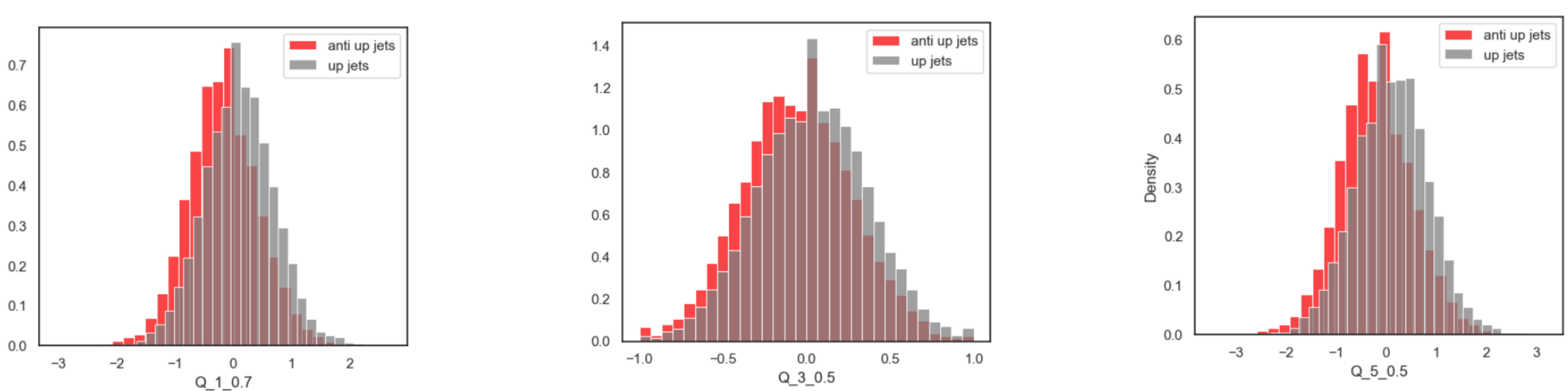
pT, eta, phi, mT, charge	Kinematic properties of 5 highest pT tracks
$Q\_1{,}2\_\kappa$	pT weighted charge
$Q\_1{,}2\_\kappa$ square	pT**2 weighted charge ($\frac{p_{Track}}{p_{jet}} > 0.1$)
$Q\_3{,}4\_\kappa$	Eta weighted Charge ($\frac{p_{Track}}{p_{jet}} > 0.1$)
$Q\_5{,}6\_\kappa$	mT weighted charge ($\frac{p_{Track}}{p_{jet}} > 0.1$)
Charge ratio	ratio of the sum of positive charges to the negative
Jet charge	sum of track's charges
Charge asymmetry	$\frac{\sum q_i \theta(\eta_i - \eta_{jet})}{\sum q_i \theta(\eta_{jet} - \eta_i)}$ where θ is the Heaviside step function

Distribution of Q variables

- Highest correlated features to the jet type are mostly Q_1_square variables.
- Q_3 and Q_5 features are less important.

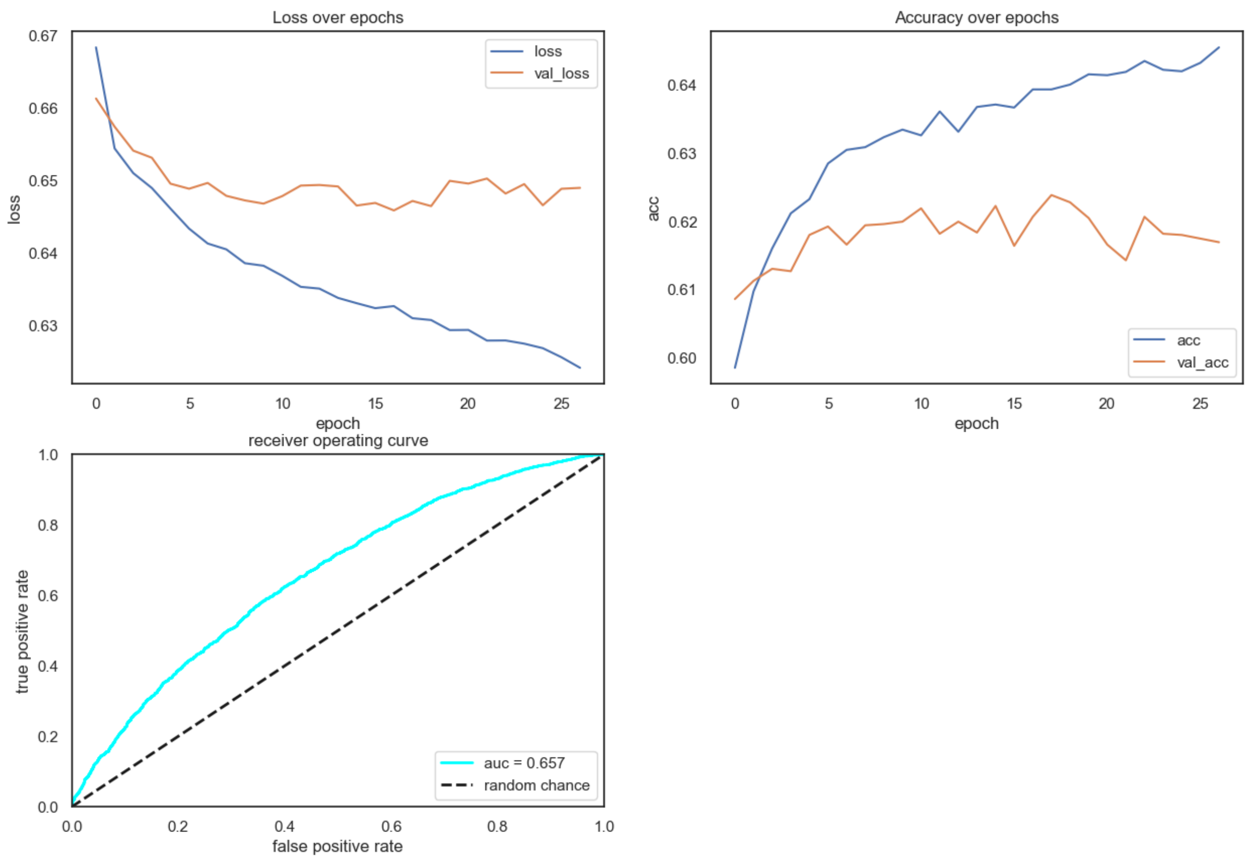


Distribution of Q-variables for different kappa values



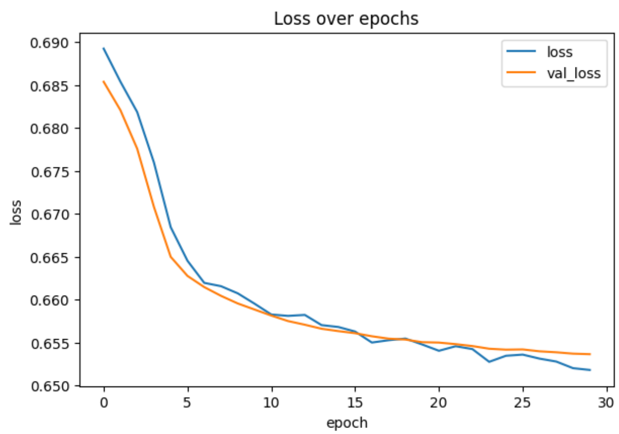
Simple DNN

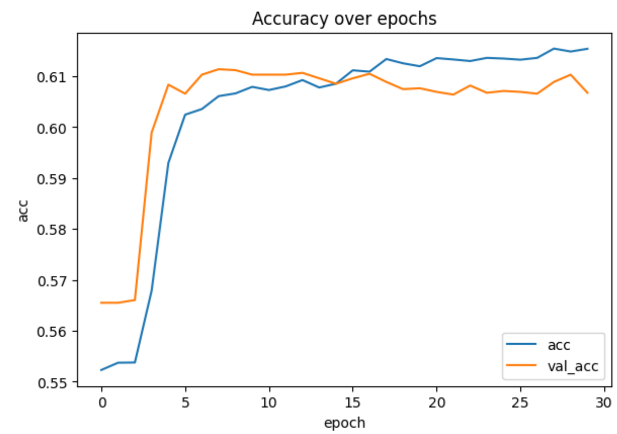
- Results for simple DNN with 2 hidden layers and 40 neurons in each. AC relu and Adam optimizer.

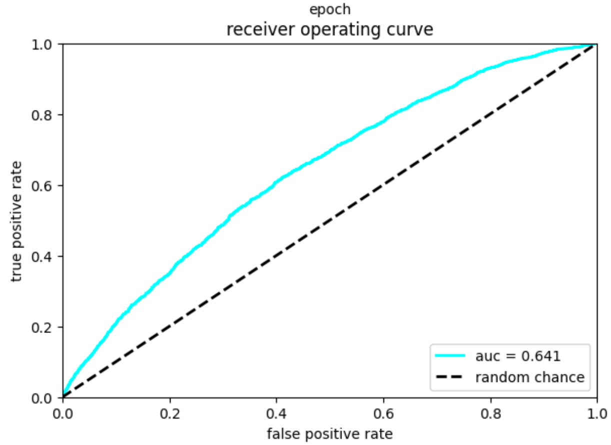


Optimized DNN

- Results for simple optimized DNN with 3 hidden layers and 32 neurons in each. AC tanh and SGD optimizer and 0.2 dropout.

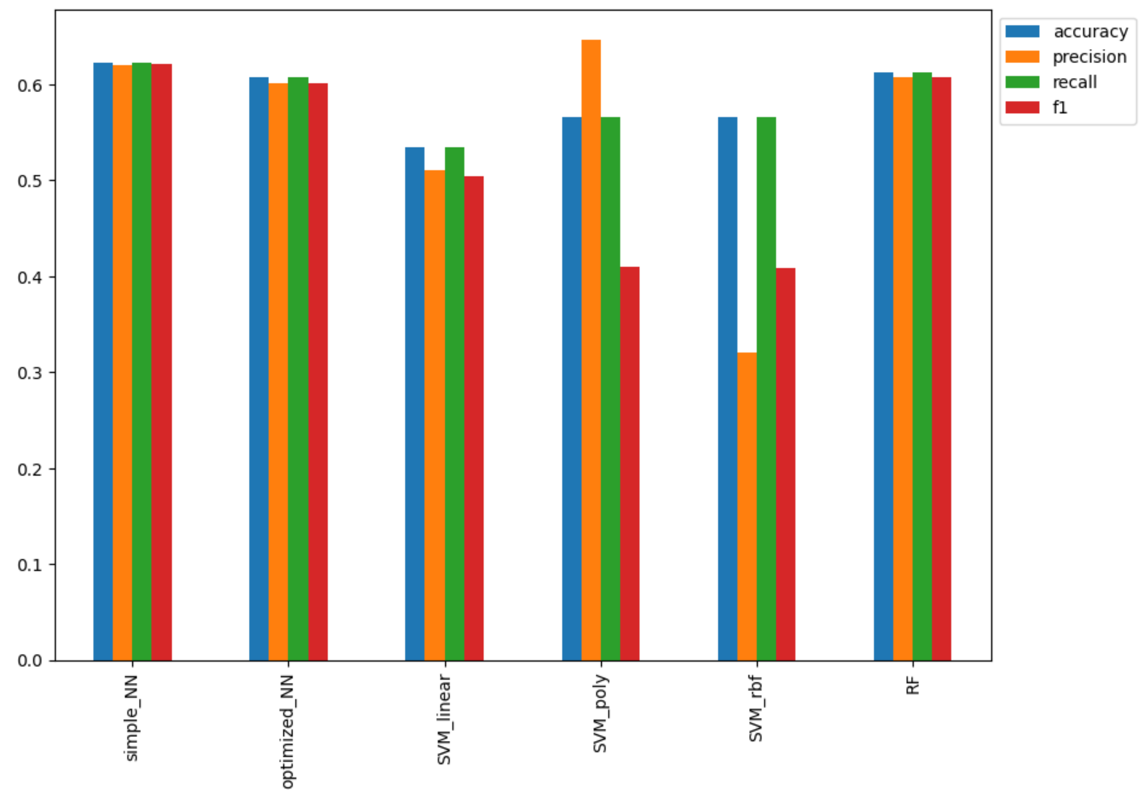






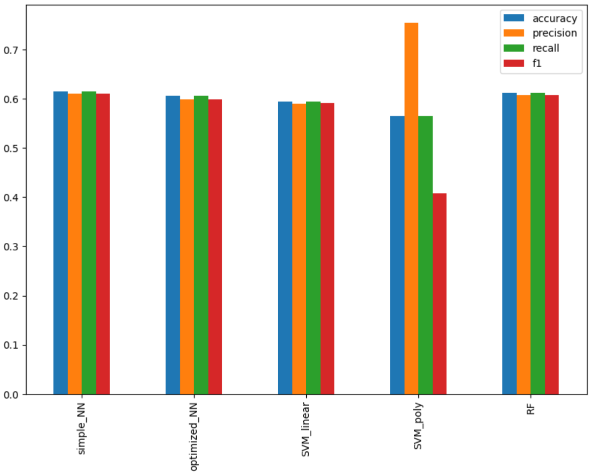
Results

- Classification metrics for well known ML models.
- NN and RF models show better performance.
- SVM with polynomial kernel achieve the highest precision (FP rate).

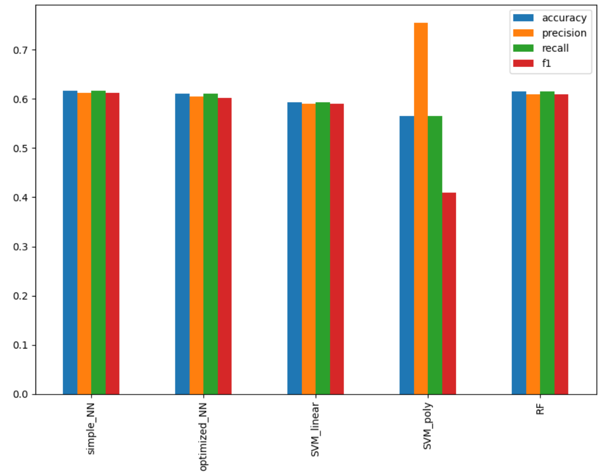


Results based on kappa

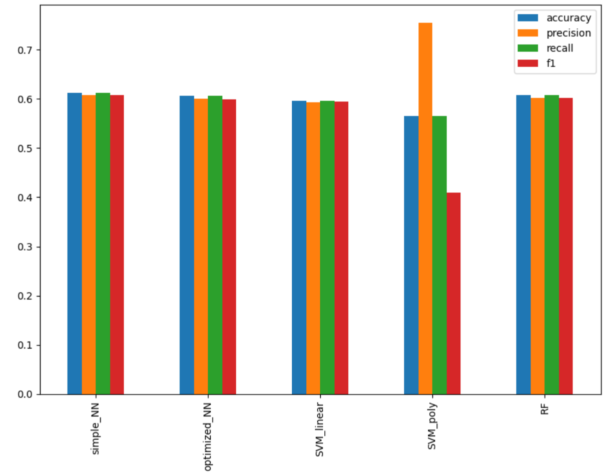
$\kappa = 0$



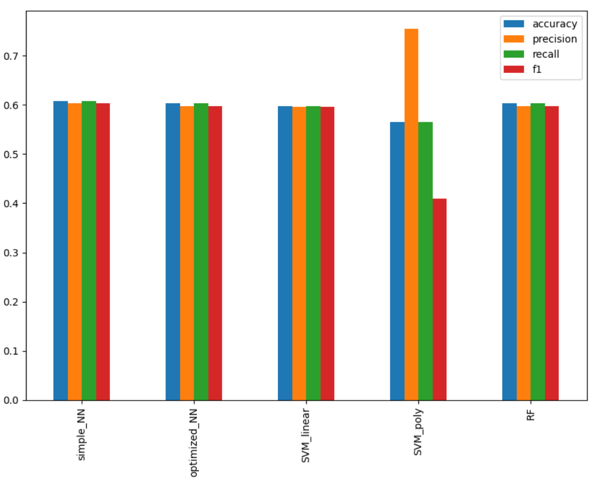
$\kappa = 0.3$



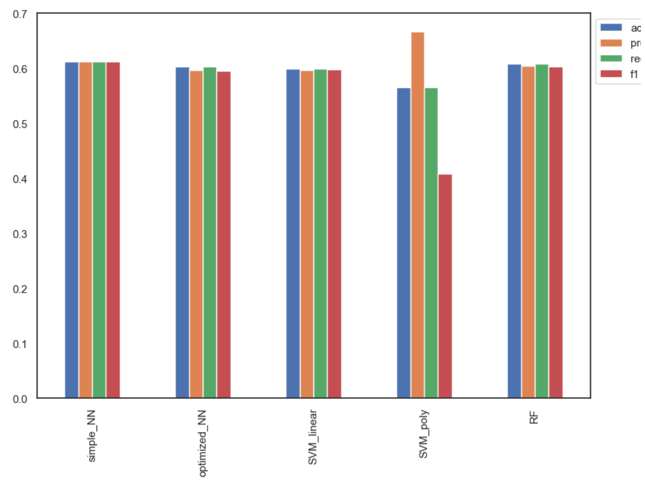
$\kappa = 0.5$



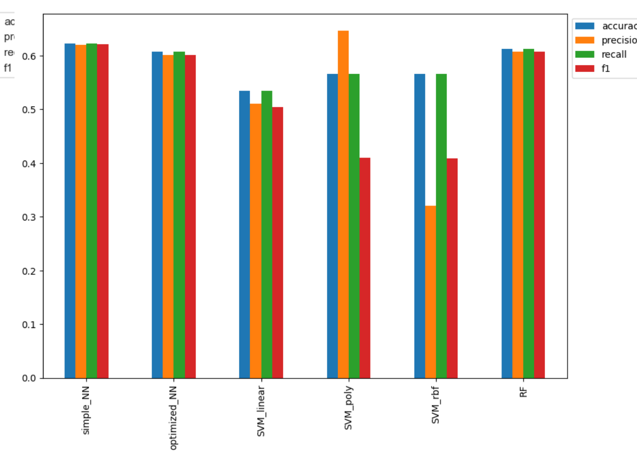
$\kappa = 0.7$



$\kappa = 1$



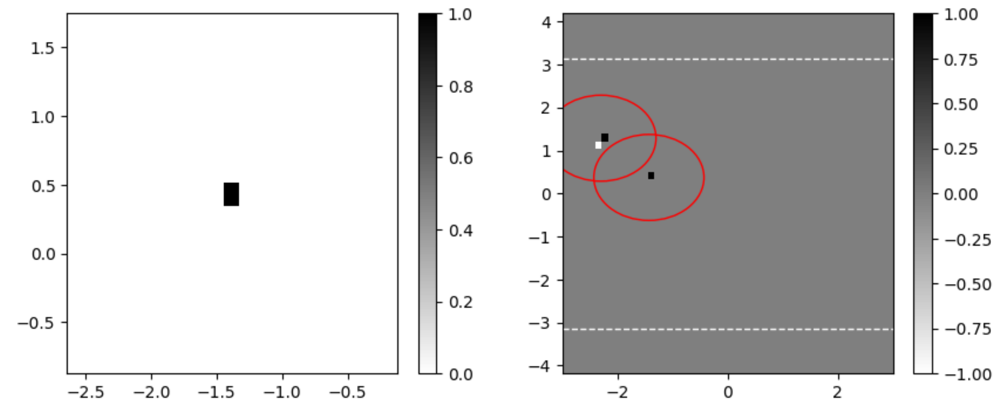
All values for κ



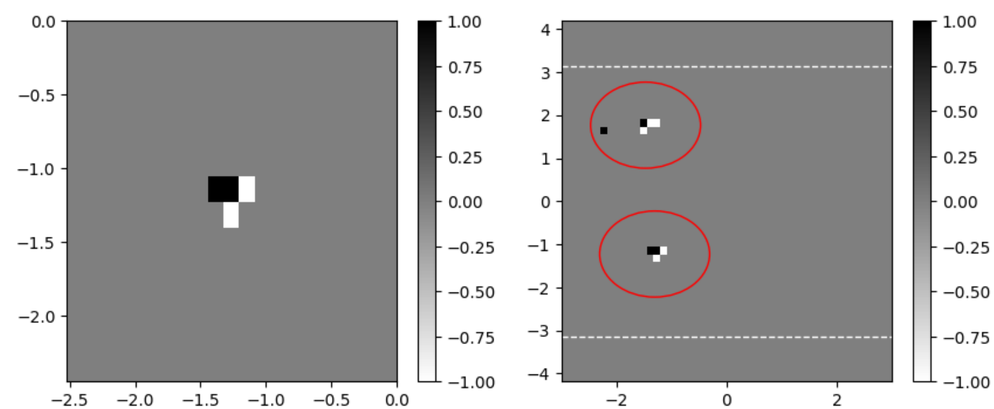
Jet images with tracks

- Leading jet image in eta and phi dimensions (left plots).
- All jet's images in eta and phi dimensions (right plots).
- There are negative tracks in up+gluon events which might come from gluon radiation (and reverse for anti up events).
- More positive tracks (black squares) in up+gluon and negative tracks (white squares) in anti up+gluon events.
- Different approach for jet classification based on charge of tracks (no more equation with kappa param).

Up jets

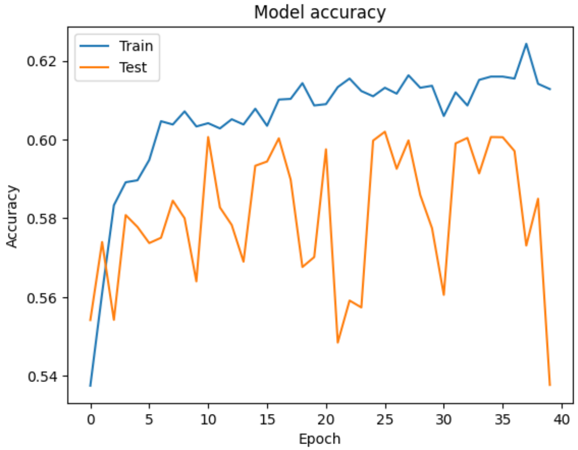


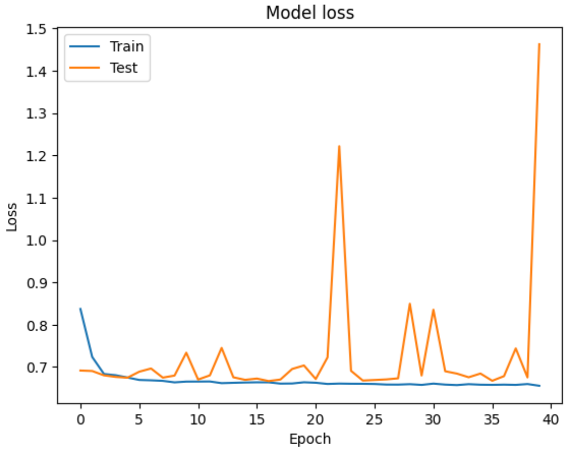
anti Up jets



CNN models and AlexNet

- CNN model with 2 conv2D, batch norm, Maxpool, and dropout layers and then flatten for binary classification (top plot).
- AlexNet with so many layers which ends up with overfitting.





Graph NN

- **Definition**: Graph Neural Networks (GNNs) are a class of neural networks designed to operate on graph-structured data.
- **Key Feature**: GNNs aggregate and transform information from a node's neighbors, leveraging the graph's connectivity.
- **Types**: Includes various models like Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs).

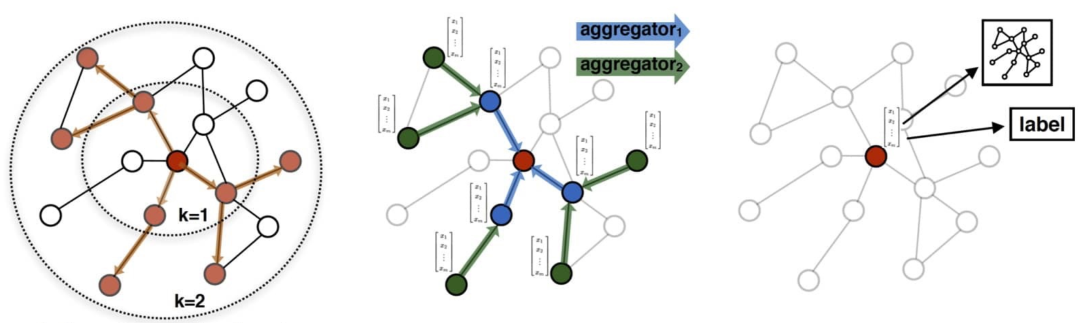
```python
# Testing function
def test():
    model.eval()
    _, pred = model(data).max(dim=1)
    correct = int(pred[data.test_mask].eq(data.y[data.test_mask]).sum().item())
    acc = correct / data.test_mask.size(0)
    return acc

# Training loop
for epoch in range(50):
    loss = train()
    if epoch % 10 == 0:
        acc = test()
        print(f'Epoch: {epoch}, Loss: {loss:.4f}, Test Accuracy: {acc:.4f}')

# Final test accuracy
acc = test()
print(f'Final Test Accuracy: {acc:.4f}')
```

```
Epoch: 0, Loss: 0.6931, Test Accuracy: 0.5447
Epoch: 10, Loss: 0.6867, Test Accuracy: 0.5447
Epoch: 20, Loss: 0.6862, Test Accuracy: 0.5447
Epoch: 30, Loss: 0.6863, Test Accuracy: 0.5447
Epoch: 40, Loss: 0.6862, Test Accuracy: 0.5447
Final Test Accuracy: 0.5447
```

Result



1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

https://github.com/mghasemi19/Jet_charge/blob/main/GNN.ipynb

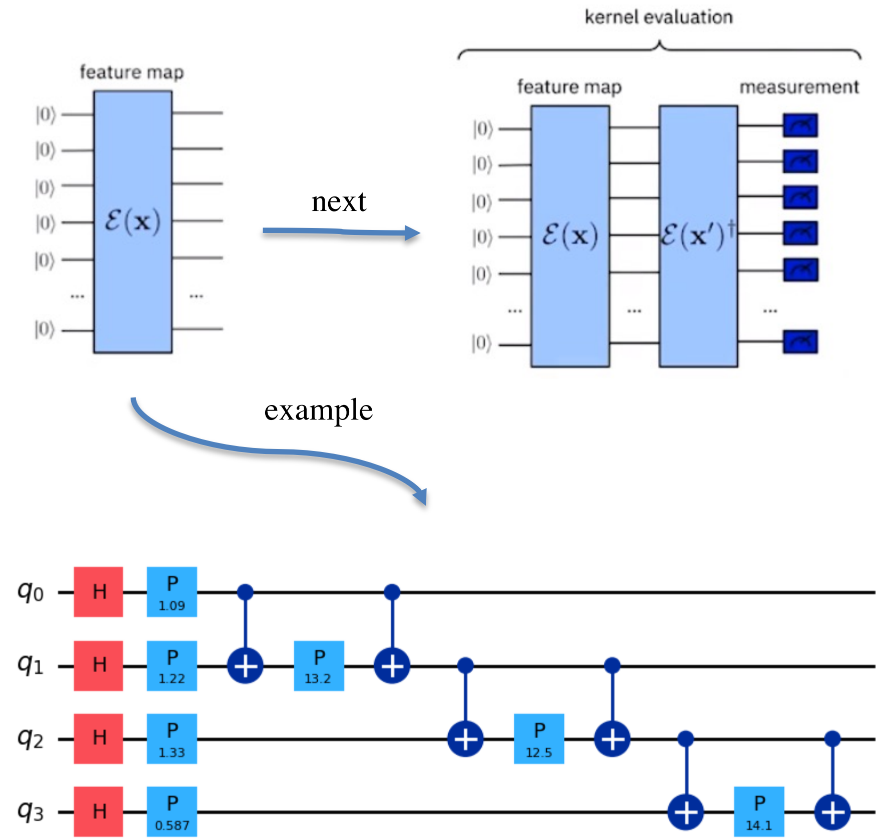
Quantum SVM

Overview:
• Combines quantum computing and classical Support Vector Machine (SVM) techniques.

Quantum Feature Map:
• Encodes classical input data into quantum states (Hilbert space)

Quantum Kernel:
• Measures similarity between quantum states of data points.
• Captures complex patterns and relationships in the data.

Advantages:
• Potential to handle large and complex datasets.
• High speed running on quantum machines with real qubits.



Result: above 90% accuracy using 1000 events.

https://github.com/mghasemi19/Jet_charge/blob/main/QSVM.ipynb

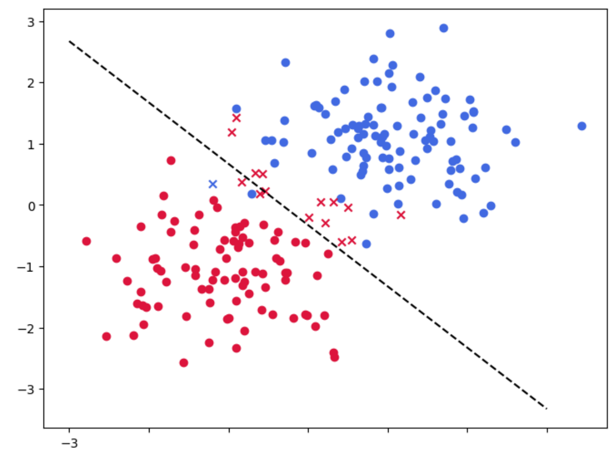
Summary & ongoing

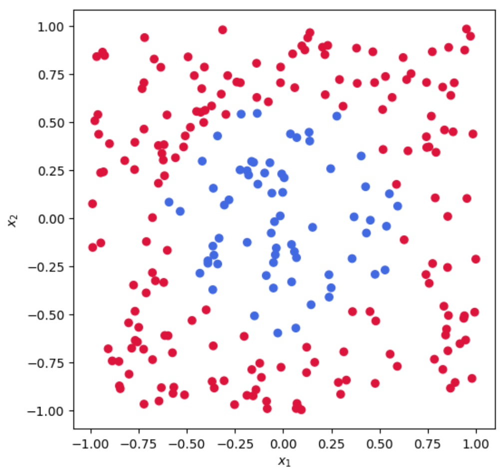
- Several variables are used to define charge of jets and fed as the inputs to ML models. pT and pT square variables show more power for classification.
- A lot of ML classifiers are trained using all/subset of data and important analysis features. NN models achieve better accuracy (63%) while SVM get the highest precision. Kappa with 0.3 value shows few percent better metrics.
- We can install Qiskit and Qiskit-machine-learning package and run with the whole dataset on IPM server.
- Analysis tree production with important variables and plotting framework are all available in notebook (link).
- Your feedback is welcome and appreciated.

Backup

Kernel in ML

Linear data





Poly kernel