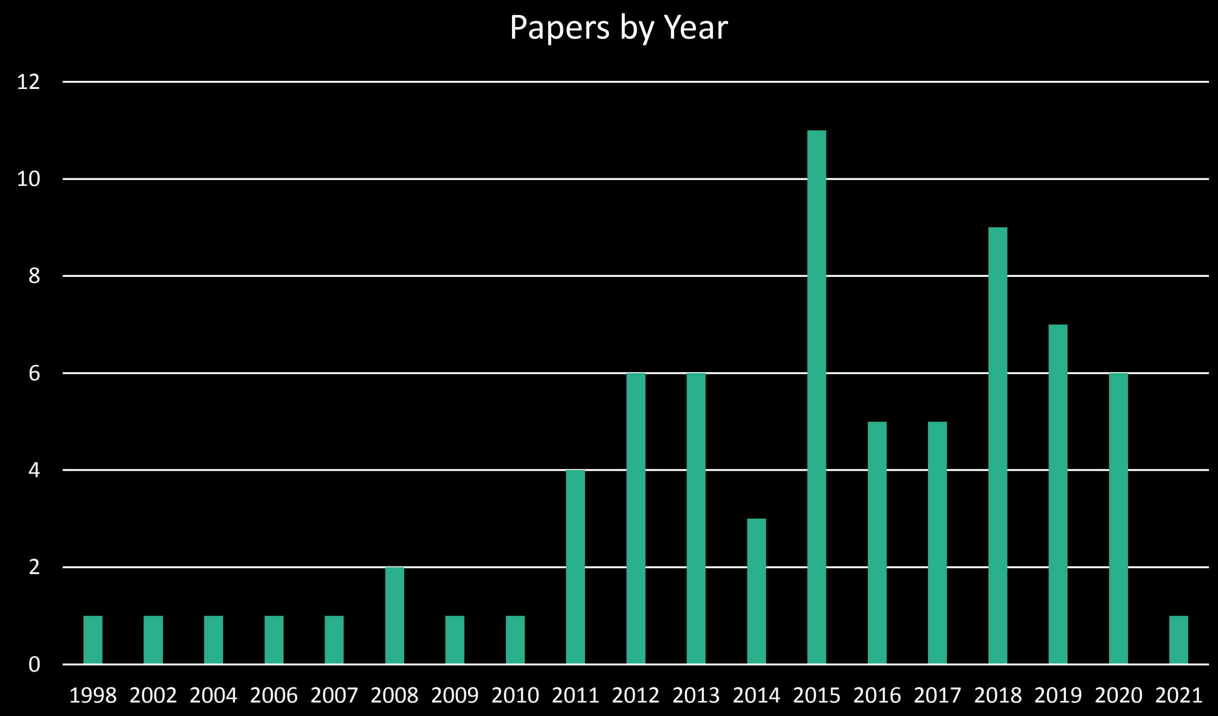
ABCD Method for LLP Searches using ML

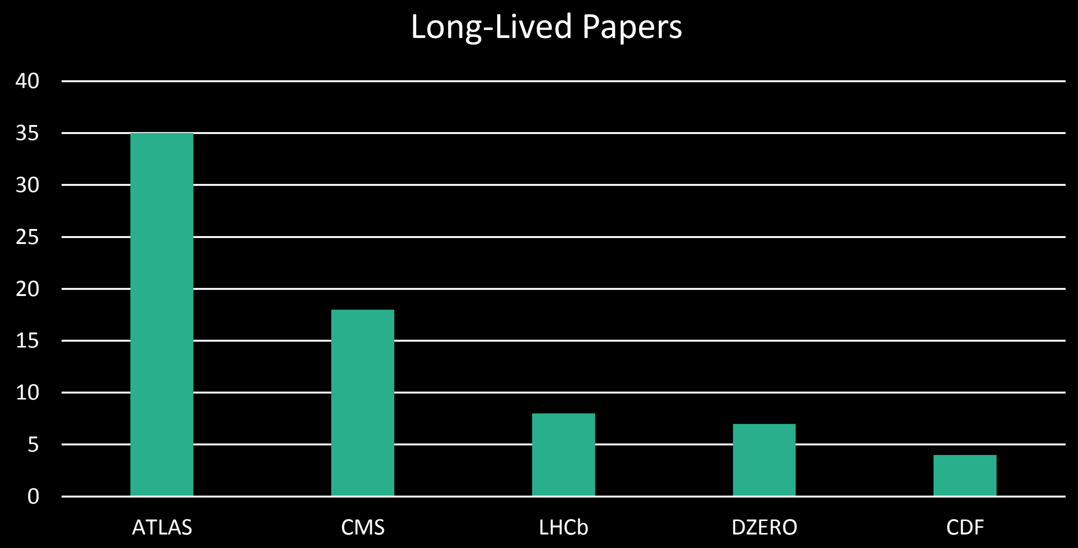
G. Watts (UW/Seattle, CPPM)

LLPX Workshop

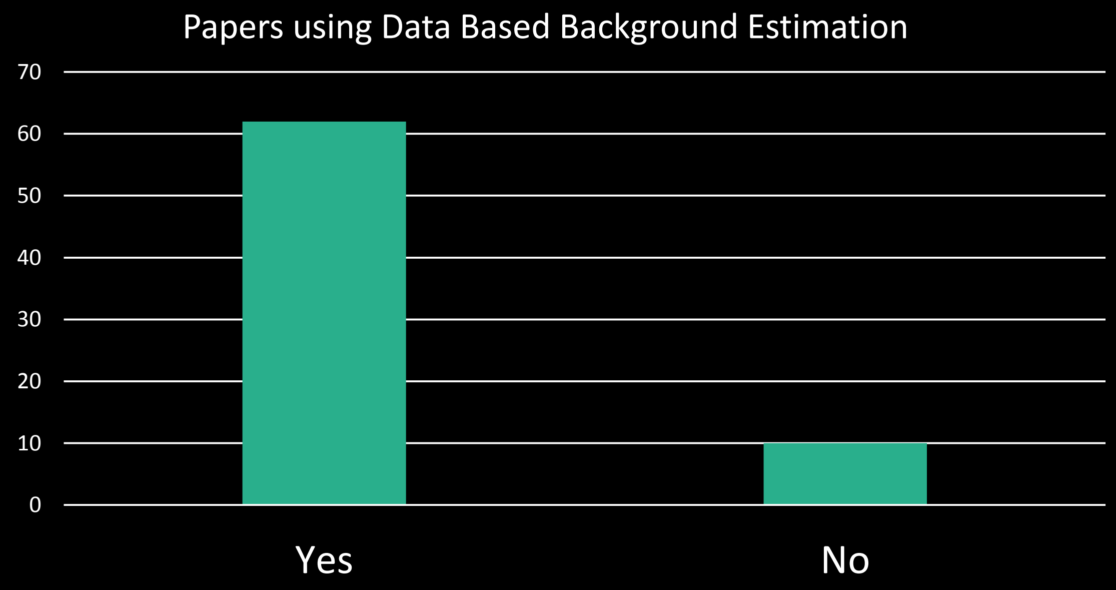
Nov 11, 2021

Background Estimation In LLP Searches





Background Estimation in LLP Searches



Many possible reasons not to trust a Monte Carlo Model

- Instrument background is hard to simulate
- Unknown physics processes
- New final state that may not be well modeled

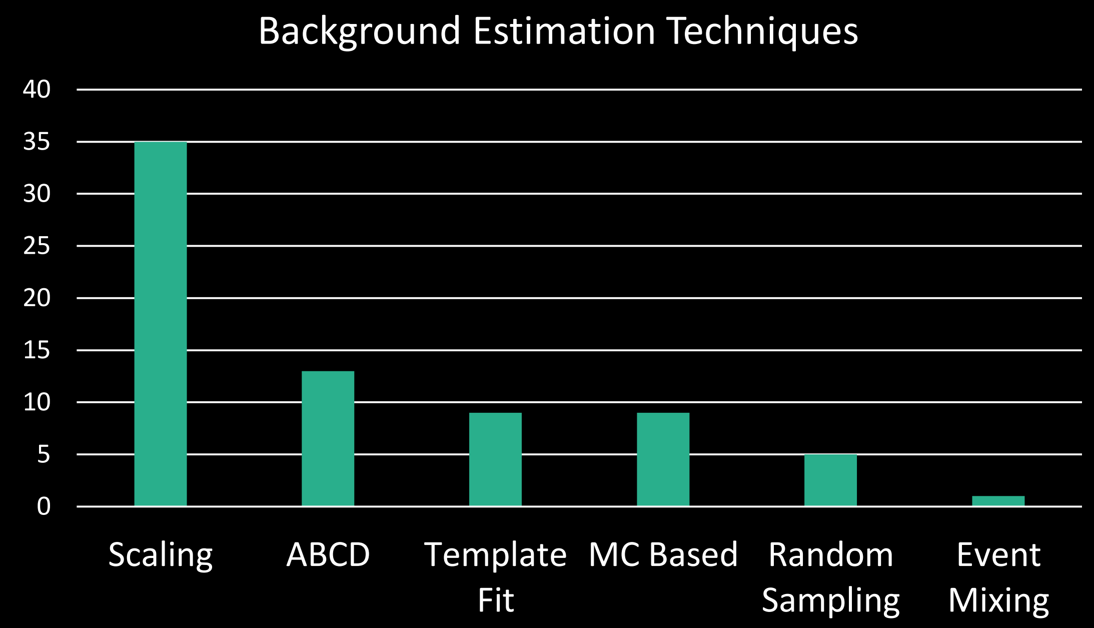
Background Estimation in LLP Searches

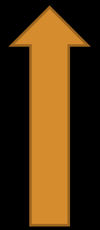
Scaling

- A Control Region exists
- A well understood scaling function exists
- Especially powerful when Control Region is high statistics

ABCD

- Two variables define a plane
- The variables are uncorrelated on sum of all backgrounds
- Signal is (mostly) confined to one quadrant
- Good when no Control Region exists





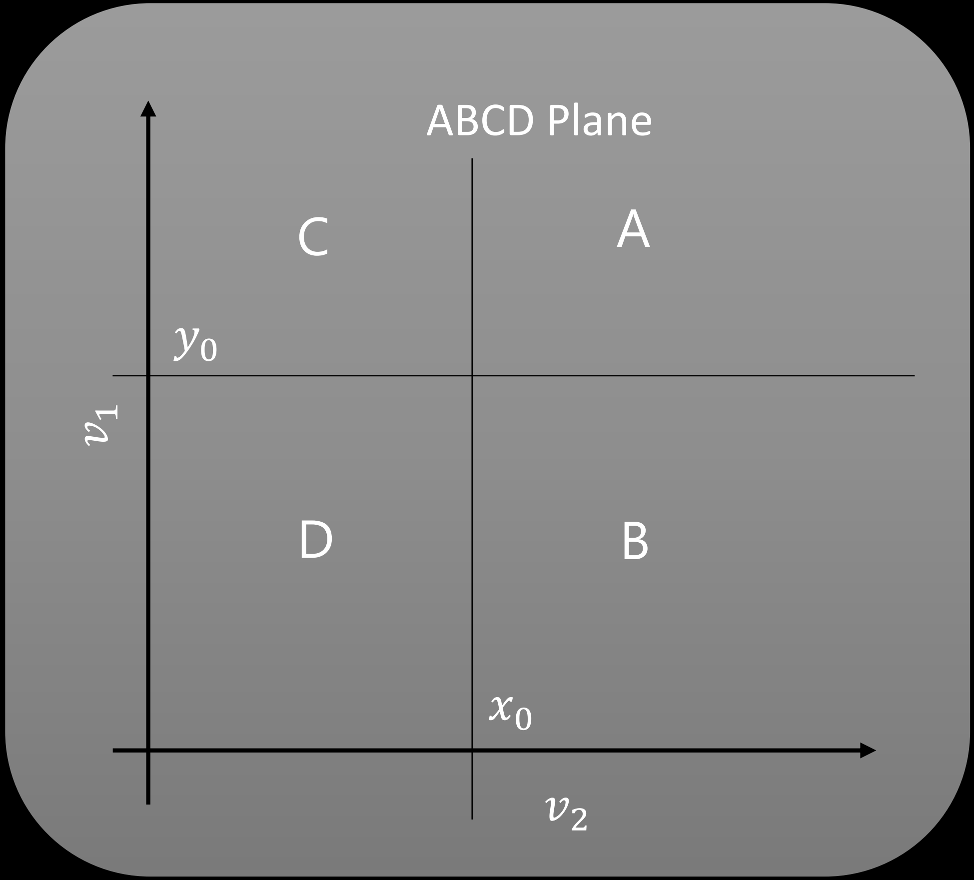
ABCD Method Refresher

$$N = A + B + C + D$$
$$= \iint f(v_1, v_2) dv_1 dv_2$$
$$= \int f_1(v_1) dv_1 \int f_2(v_2) dv_2$$

Since f is uncorrelated, $f = f_1 f_2$

$$A = \int_{y_0} f_1(v_1) dv_1 \int_{x_0} f_2(v_2) dv_2 , B = \cdots , C = \cdots , D = \cdots$$

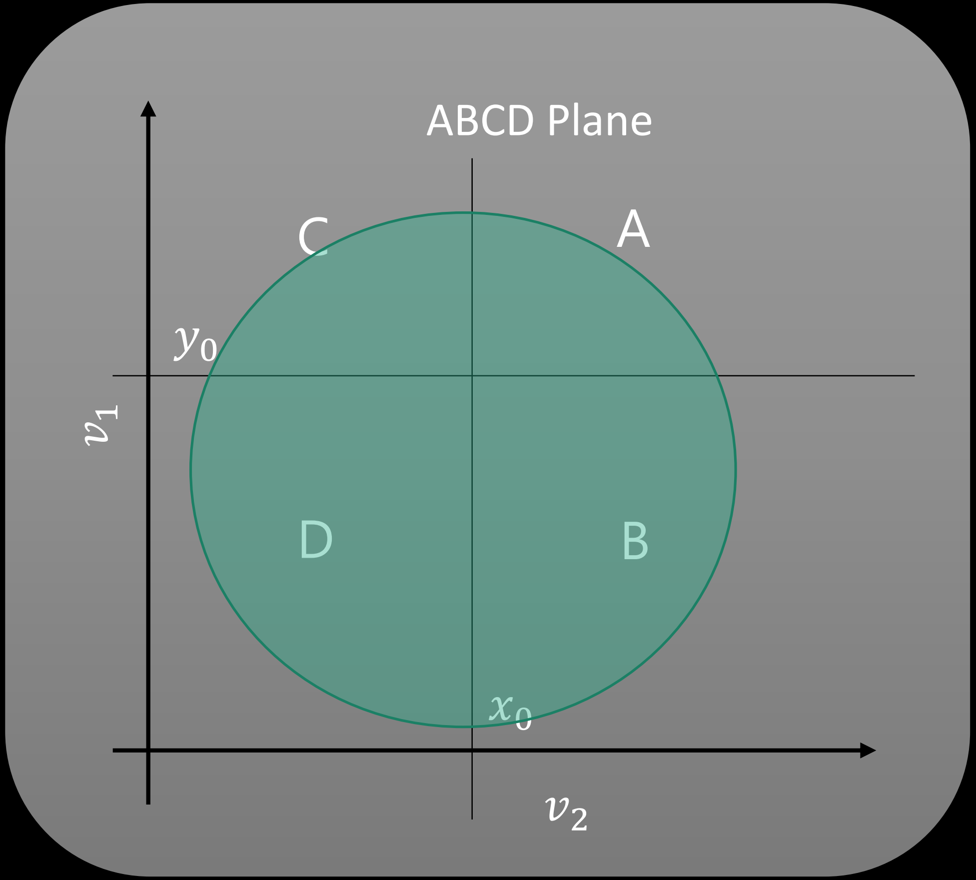
$$\Rightarrow \frac{A}{B} = \frac{C}{D}$$



ABCD Method Refresher

$$\frac{A}{B} = \frac{C}{D}$$

1. Your background data is distributed over the $ABCD$ plane
2. Your signal is confined to region A
3. Your expected background in region $A = CB/D$

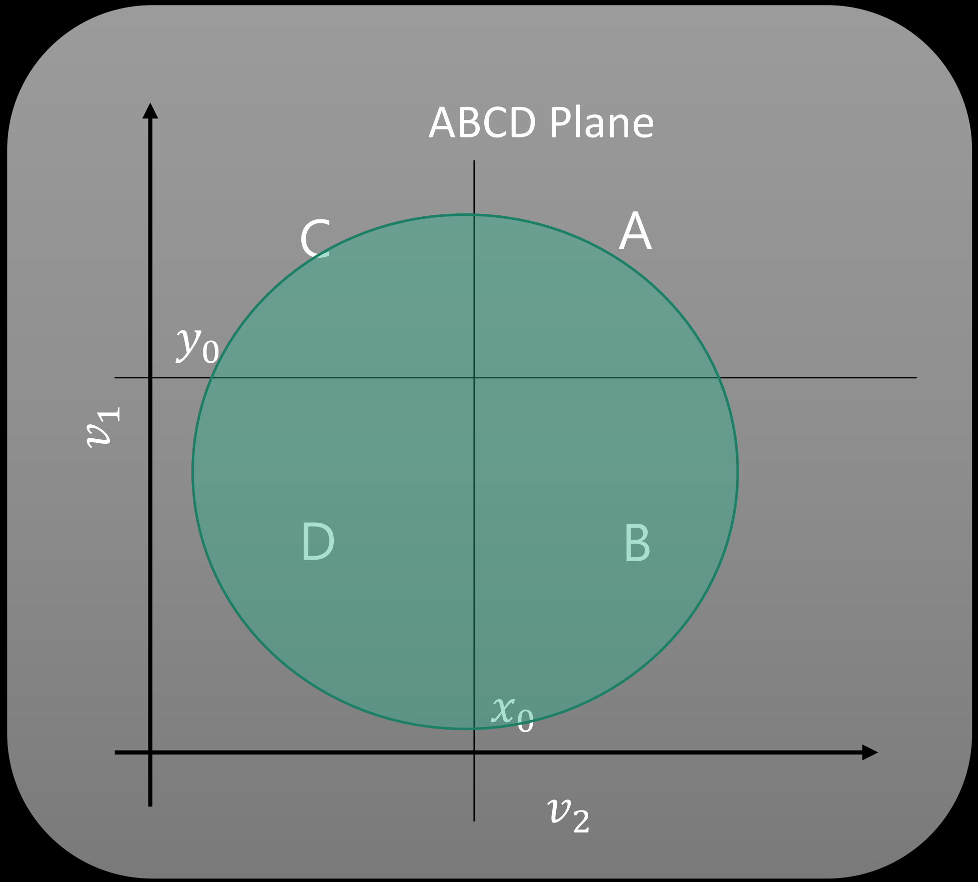


ABCD Method Refresher

$$\frac{A}{B} = \frac{C}{D}$$

1. Your background data is distributed over the $ABCD$ plane
2. Your signal is confined to region A
3. Your expected background in region $A = CB/D$

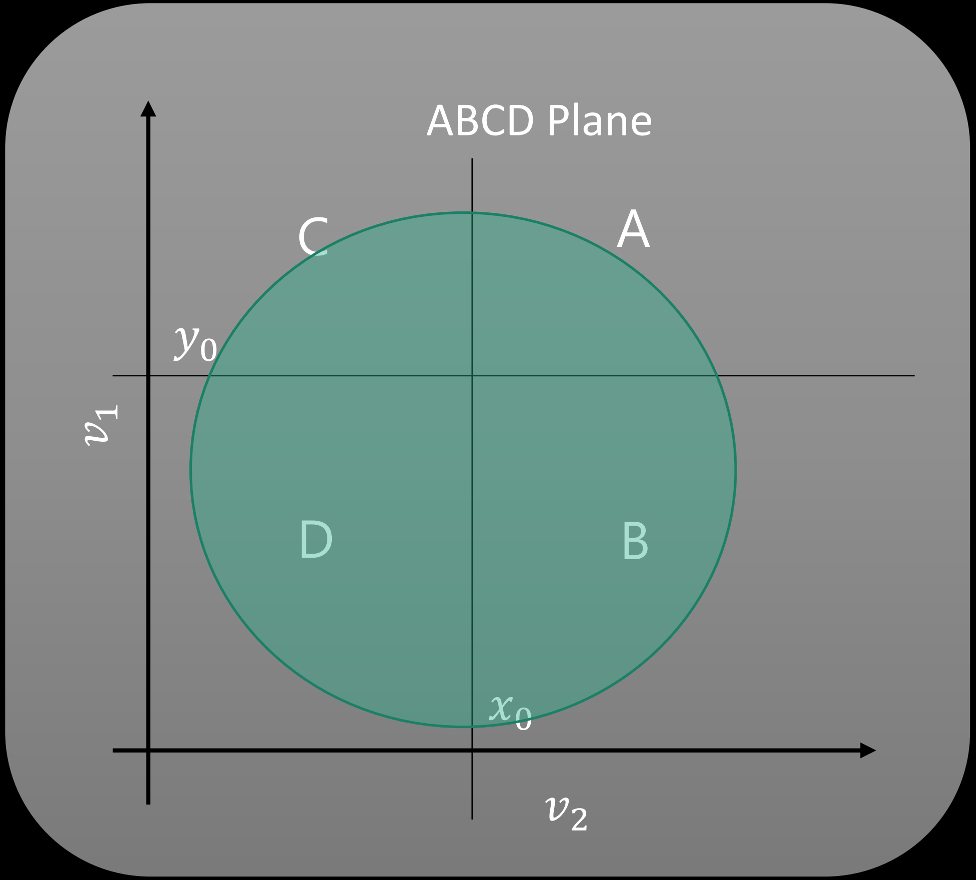
- Potential Issues:
  - Signal may leak out of region A
  - Multiple Backgrounds
  - Statistics
  - Validation



Axes

From the quick paper survey:

- Lepton Quality Cuts
- $\Delta R(track, jet)$
- $\Delta\phi(jets)$
- Boosted Decision Tree output
- dE/dx
- Lepton Isolated E_T



- Machine Learning appears infrequently!
- Some selection items are binary

Modified ABCD Method

Signal leakage is a real problem!

Leakage of about 10% outside A is probably tolerable

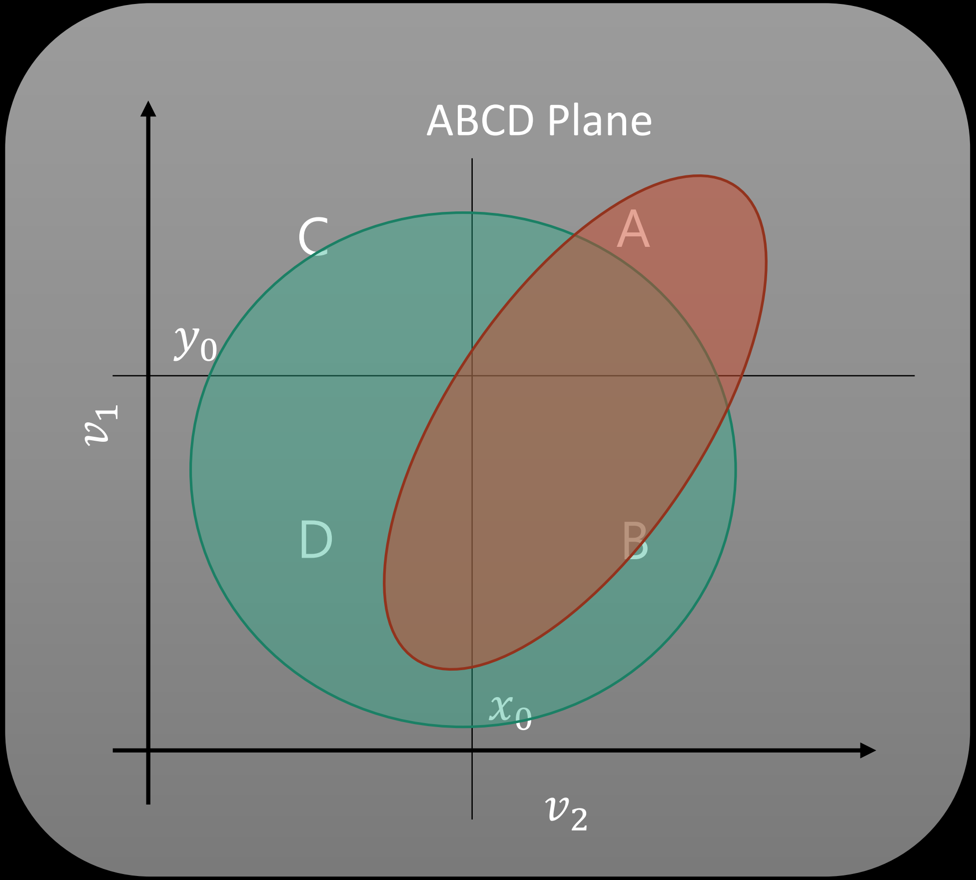
Finding uncorrelated axes with real separation power is difficult

- We are probing rare and difficult to find signals
- Rarely we have a single, good, handle/variable any longer

$$
\begin{aligned}
A &= A_{back} + A_{sig} \\
B &= B_{back} + B_{sig} \\
C &= C_{back} + C_{sig} \\
D &= D_{back} + D_{sig}
\end{aligned}
\qquad
A_{back} = \frac{B_{back} C_{back}}{D_{back}}
$$

Fit using signal shape and tool like pyHF or RooFit

Would be better not to…



Implementation of the likelihood-based ABCD method for background estimation and hypothesis testing with pyhf (upcoming poster at ACAT 2021 by Mason Proffitt)

Adding Machine Learning

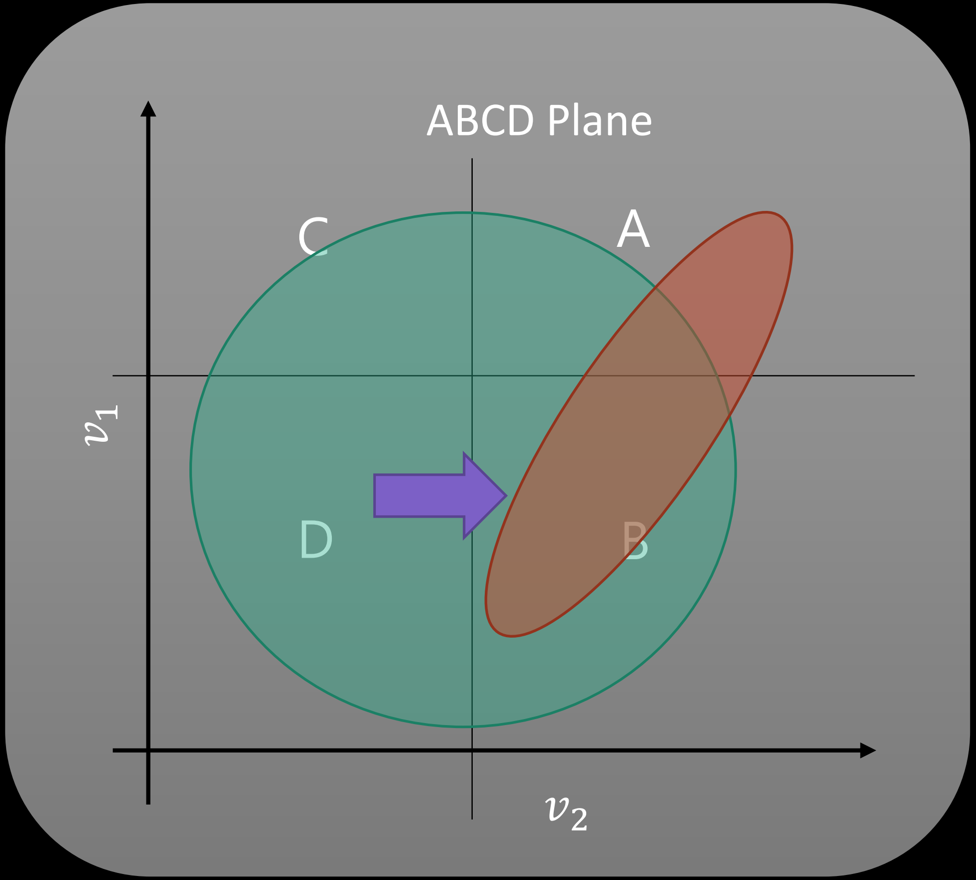
1. Train a ML algorithm to score signal vs background
2. Use it as one of the axes

Likely to push signal further into one of the half planes

In CalRatio in our last publication:

- *"Simple"* BDT #1: separate displaced jets from SM jets
- "Simple" BDT #2: Event topology including inputs from #1, trained to remove BIB, and separate signal from background
- #2 was used as one of the axes
- A ΔR variable was the second axis

Achieved between 15-20% improvement in acceptance

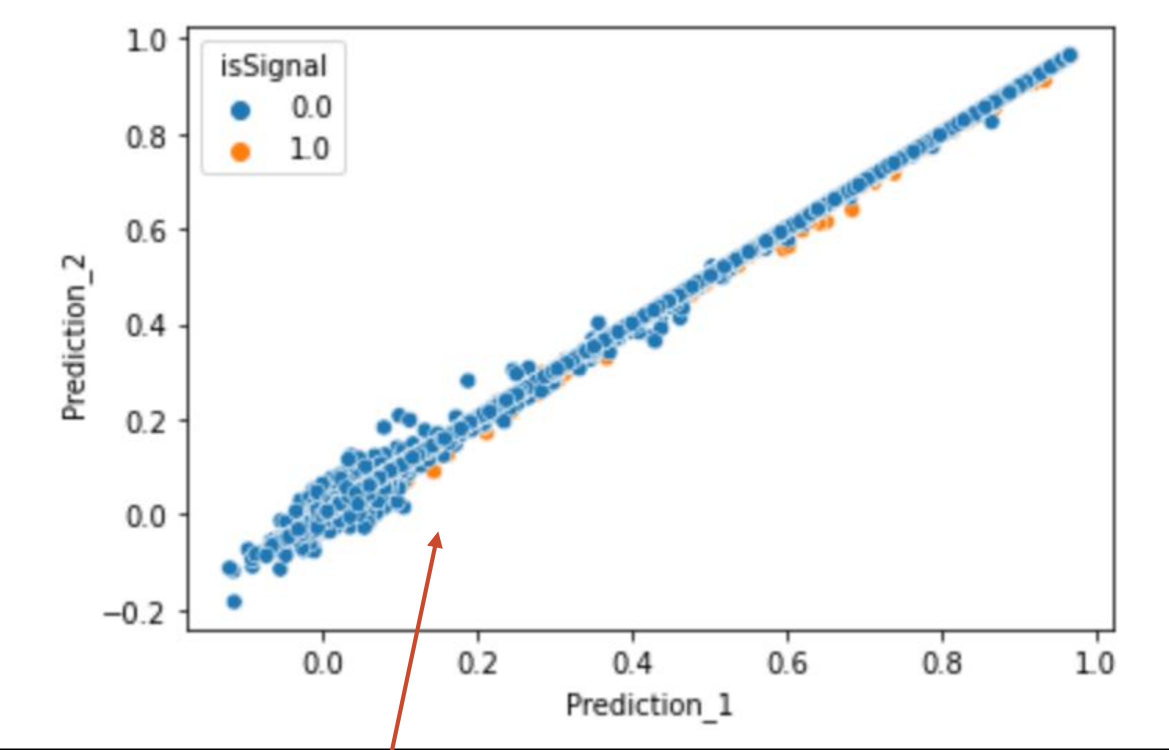


Use ML for Both Axes?

Possible Approaches

- Divide variables into two uncorrelated groups, train separate ML's
- Train a single ML with two outputs, somehow demand decorrelation

Attractive: can split the separation power between both variables evenly, making the ABCD plane better behaved (statistically).



But the laziest thing for a network to learn is to have the outputs mirror each other!

ML is not a black box

PyTorch (and TF)

- Allows you to modify all steps of the training…
- As long as gradient's can be calculated (forward and backwards)

```
[7]: epochs = 5000
for e in range(epochs):
    running_loss = 0
    optimizer.zero_grad()
    output = model(x_train)
    loss = criterion(output, y_train)
    loss.backward()
    optimizer.step()
    running_loss += loss.item()
else:
    print(f'Training loss: {running_loss/len(x_train)}')

Training loss: 1.1082058399915695e-05
```

```
model = nn.Sequential(nn.Linear(n_variables, n_variables*2),
                      nn.ReLU(),
                      nn.Linear(n_variables*2, n_variables),
                      nn.ReLU(),
                      nn.Linear(n_variables, 2))
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)
```

MSELOSS

CLASS `torch.nn.MSELoss(size_average=None, reduce=None, reduction: str = 'mean')` [SOURCE]

Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input x and target y.

The unreduced (i.e. with `reduction` set to `'none'`) loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = (x_n - y_n)^2,$$

Modify the Loss Function

1. Separation between signal and background
2. Uncorrelated on background

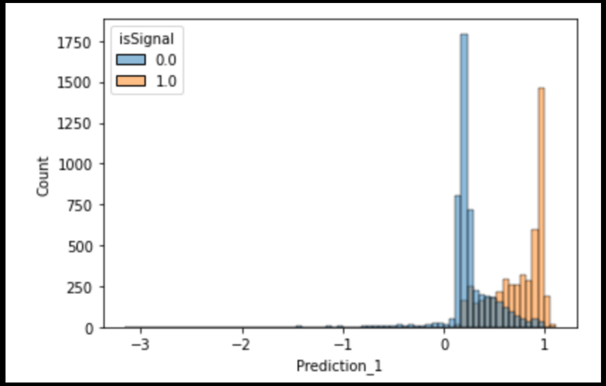
MSELoss gives us this by comparing with *ground truth* in the training (this is supervised training, after all)

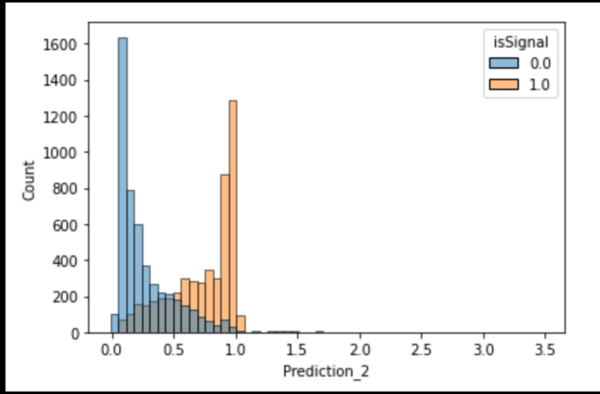
Technically: we want ***r*** (correlation coefficient) to be zero.

- r is both positive and negative, depending
- Use r^2 instead
- This adds a penalty for any correlation in the data!

Add Person Correlation Coefficient…

```python
def calc_r(prediction):
    mean = torch.mean(prediction, dim=0)
    std_dev = torch.std(prediction, dim=0)
    parts = (prediction - mean)
    sum = torch.sum(parts[:,0]*parts[:,1])
    return sum / std_dev[0] / std_dev[1] / (prediction.shape[0]-1)

class decorrelate_loss:
    '''Calculate the loss function using MSELoss and decorrelation loss
    '''
    def __init__(self):
        self._mse = nn.MSELoss(reduction='mean')

    def __call__(self, prediction, labels):
        'Calc the loss given both the correlation and mse'
        mse_loss = self._mse(prediction, labels)

        background_mask = labels[:,1] == 0
        r = calc_r(prediction[background_mask])

        total = mse_loss + torch.square(r)*0.1
        return total
```



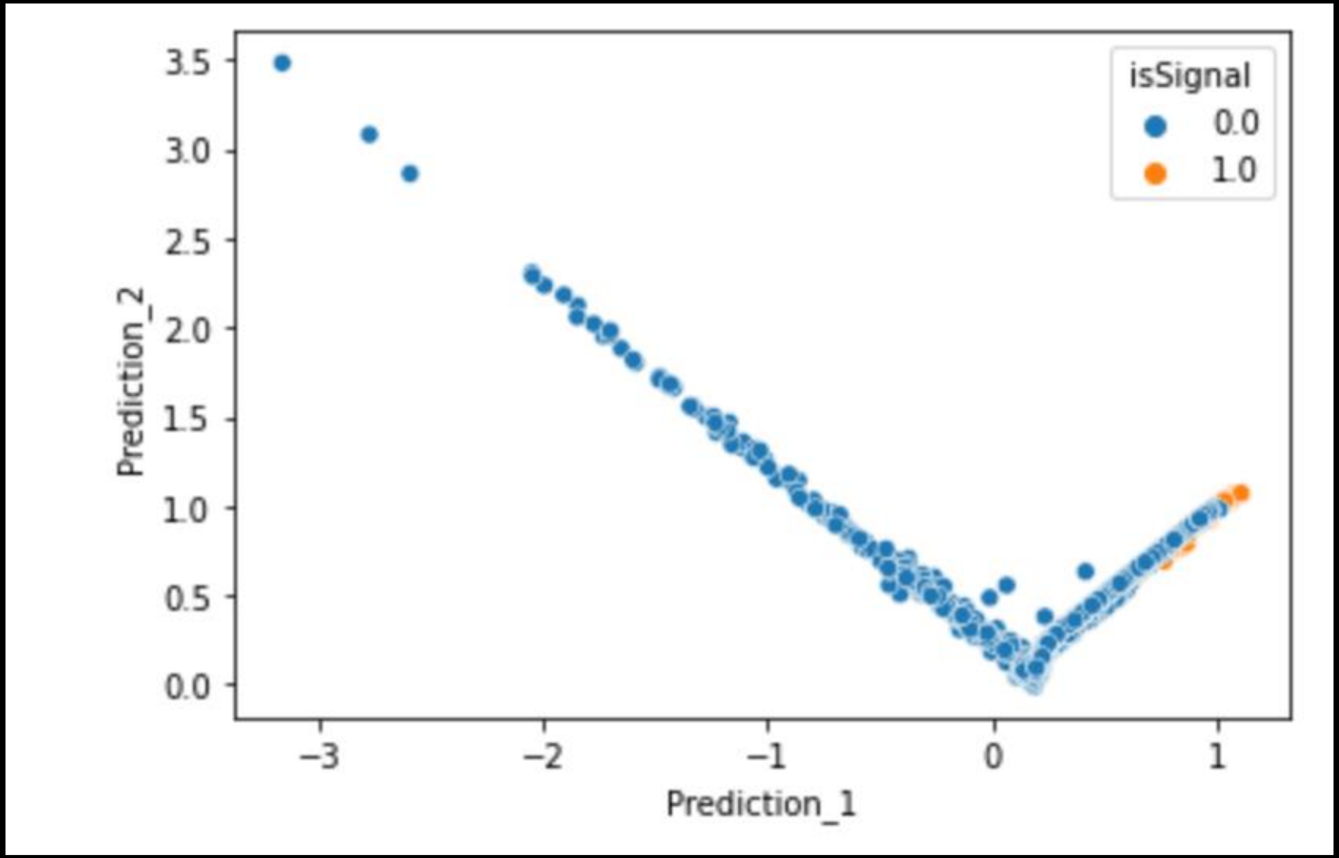



```python
label = torch.Tensor(testing[testing.columns[-1]].values)
mask = label == 0.0
calc_r(y_test[mask])
```

```
tensor(0.0349, grad_fn=<DivBackward0>)
```

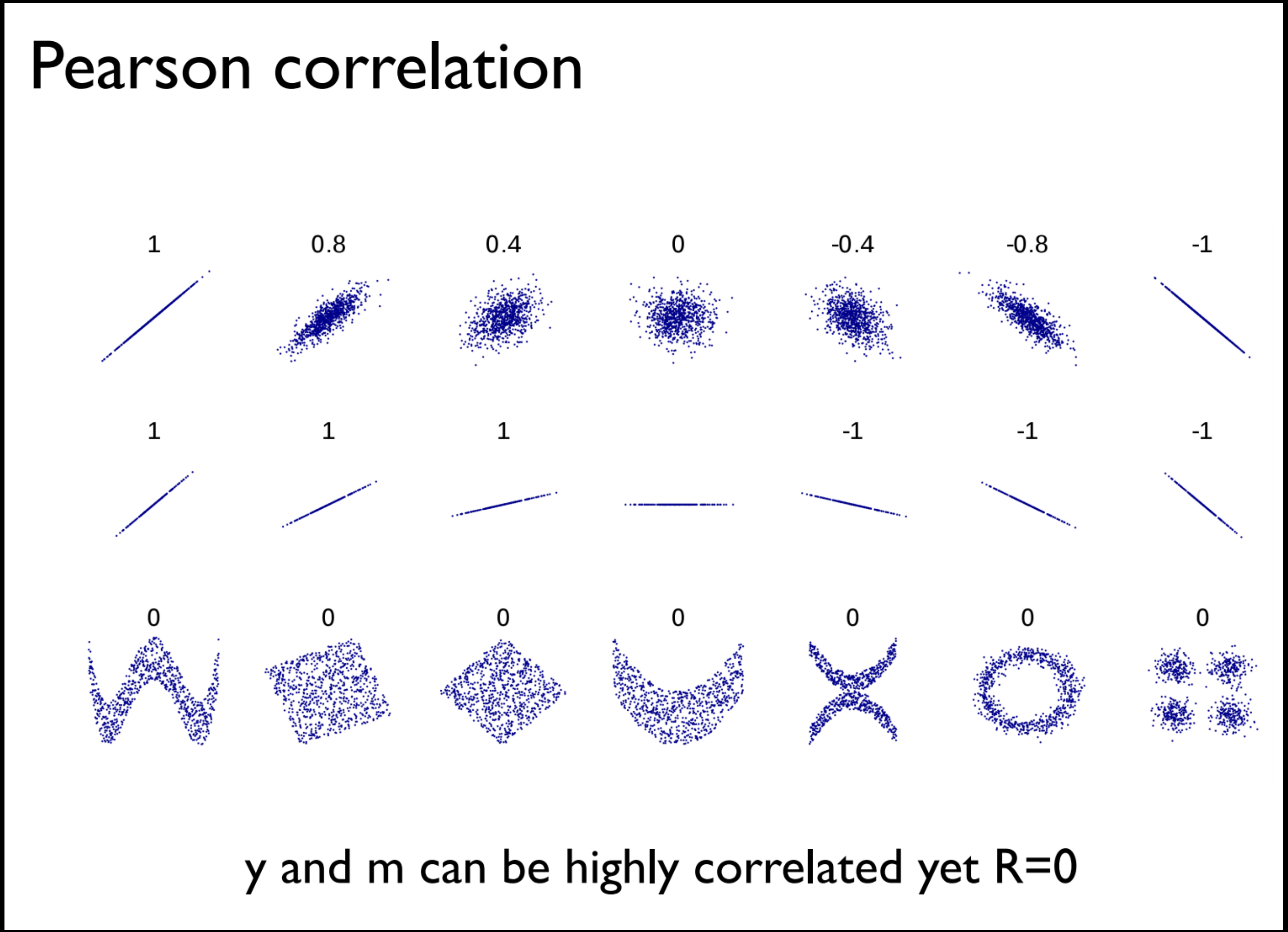
Correlation nearly zero!!
But what are those tails?

Well… it did do what we told it to do…

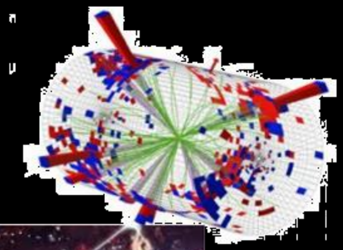


So trivial to add unintended biases… No wonder ML gets a bad name…

From the DisCo talk…



Need a better term... Distance Correlation

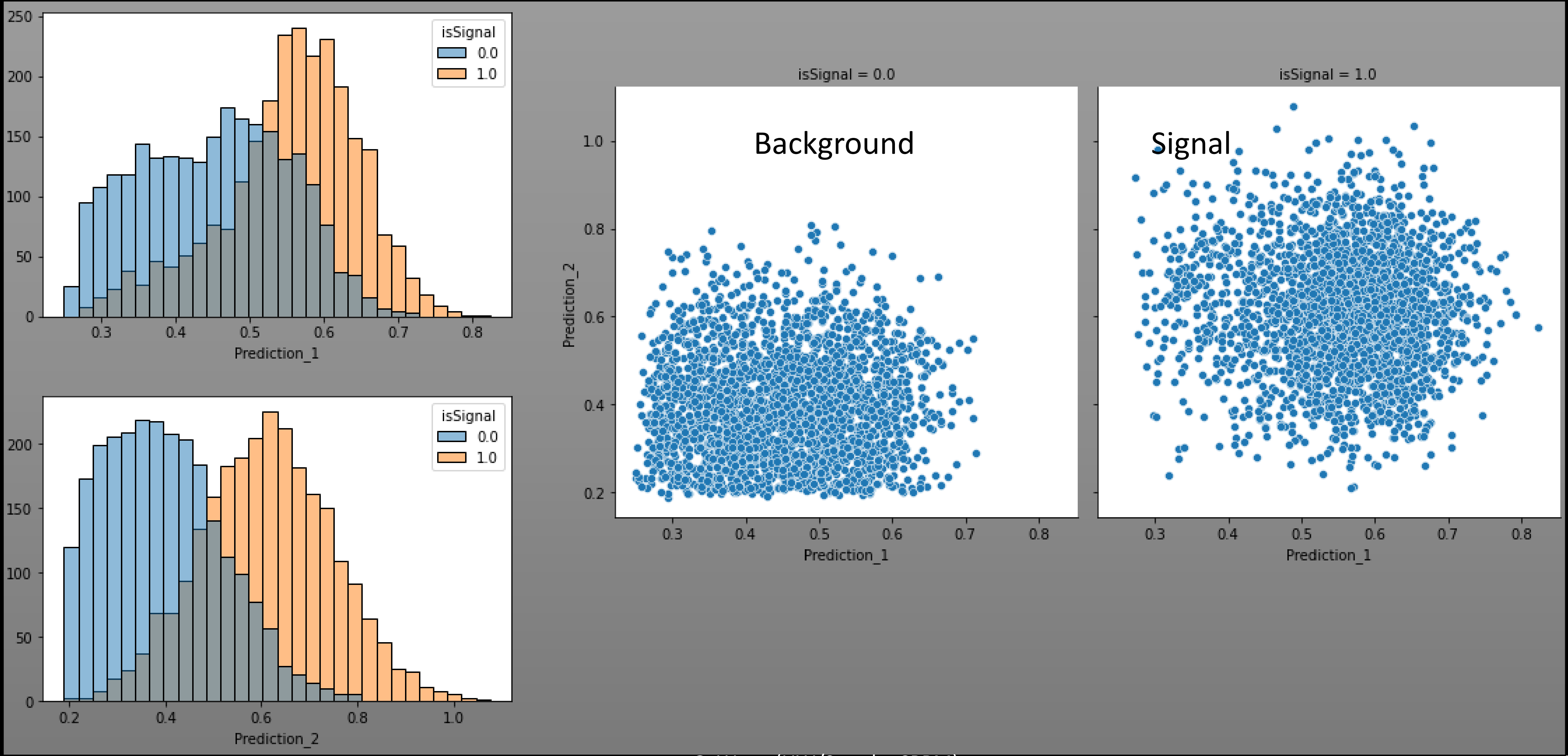


$$
\begin{aligned}
dCov^{2(X,Y)} &= \langle |X - X'||Y - Y'| \rangle \\
&+ \langle |X - X'| \rangle \langle |Y - Y'| \rangle \\
&- 2 \langle |X - X'||Y - Y'| \rangle
\end{aligned}
$$

- Zero iff X,Y are statistically Independent
- Positive Otherwise
- Tractable in ML trailing and gradient calculations!

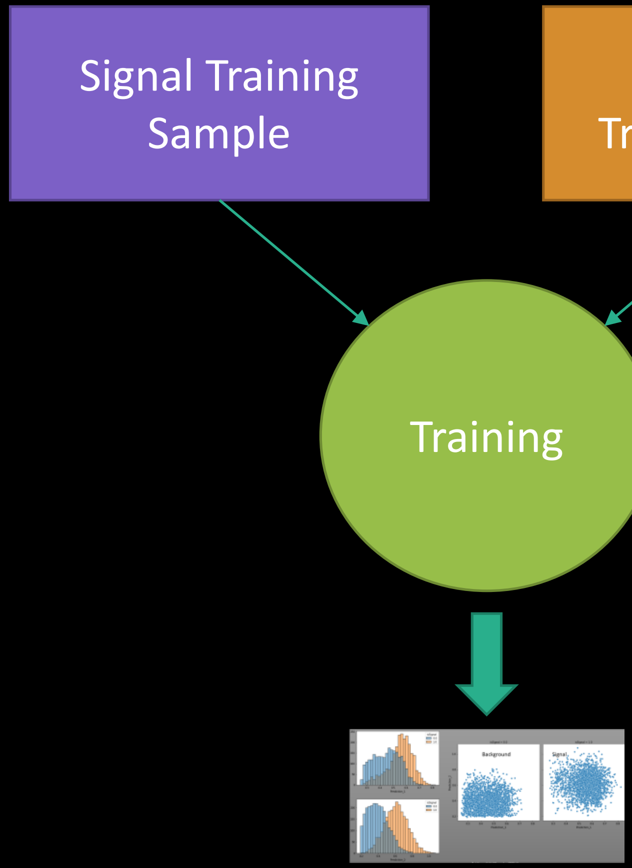
Distance Correlation Term: Szekely, Rizzo, Bakirov 2007, Szekely & Rizzo 2009
DisCo Fever (ml usage): G. Kasieczka & D. Shih, PRL 125 (2020), 2001.05310
ABCDisCo (usage): G. Kasieczka, B. Nachman, D. Schwartz, D. Shih, Phys. Rev. D 103, 035021 (2021) 2007.14400

Ahhh….



isSignal
0.0
1.0
Prediction_1
Prediction_2
isSignal = 0.0
isSignal = 1.0
Background
Signal

Still a long way to go…

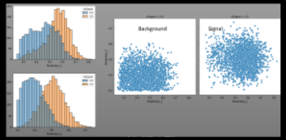


Still a long way to go…

Signal Training Sample

Background Training Sample

Training



- Where do you get a background model for the full ABCD plane?
- We wouldn't be in this situation if we had an accurate background model here!

Still a long way to go…

Background Training Sample

- Good Enough Background Model
  - Correlation is ok
  - Separation with signal is ok
- Inaccuracies will show up as reduced acceptance

The ABCD method is 100% data driven!

Conclusion

- The ABCD method has been with us since before the Tevatron
  - Any analysis with a poorly simulated background model is a candidate
  - Like many LLP analyses
  - Shines when background can't be scaled from a high statistics Control Region
- Machine Learning is already improving ABCD's effectiveness
- The DisCo method is a more automated way to approach the ABCD method
  - As long as you have the training samples
  - And can provide the validation
- What is next?
  - Use the sensitivity for signal, including systematic errors, to help drive the training!
  - With this you could drive the x_0, y_0 determination as well.