# Duarte-hands-on2

February 16, 2026

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb

%matplotlib inline
```

[2]:
```
data = pd.read_csv("data/training_10k.csv")
```

[3]:
```
print("Size of data: {}".format(data.shape))
print("Number of events: {}".format(data.shape[0]))
print("Number of columns: {}".format(data.shape[1]))

print("\nList of features in dataset:")
for col in data.columns:
    print(col)
```

```
Size of data: (10000, 33)
Number of events: 10000
Number of columns: 33

List of features in dataset:
EventId
DER_mass_MMC
DER_mass_transverse_met_lep
DER_mass_vis
DER_pt_h
DER_deltaeta_jet_jet
DER_mass_jet_jet
DER_prodeta_jet_jet
DER_deltar_tau_lep
DER_pt_tot
DER_sum_pt
DER_pt_ratio_lep_tau
DER_met_phi_centrality
DER_lep_eta_centrality
PRI_tau_pt
PRI_tau_eta
```

```
PRI_tau_phi
PRI_lep_pt
PRI_lep_eta
PRI_lep_phi
PRI_met
PRI_met_phi
PRI_met_sumet
PRI_jet_num
PRI_jet_leading_pt
PRI_jet_leading_eta
PRI_jet_leading_phi
PRI_jet_subleading_pt
PRI_jet_subleading_eta
PRI_jet_subleading_phi
PRI_jet_all_pt
Weight
Label
```

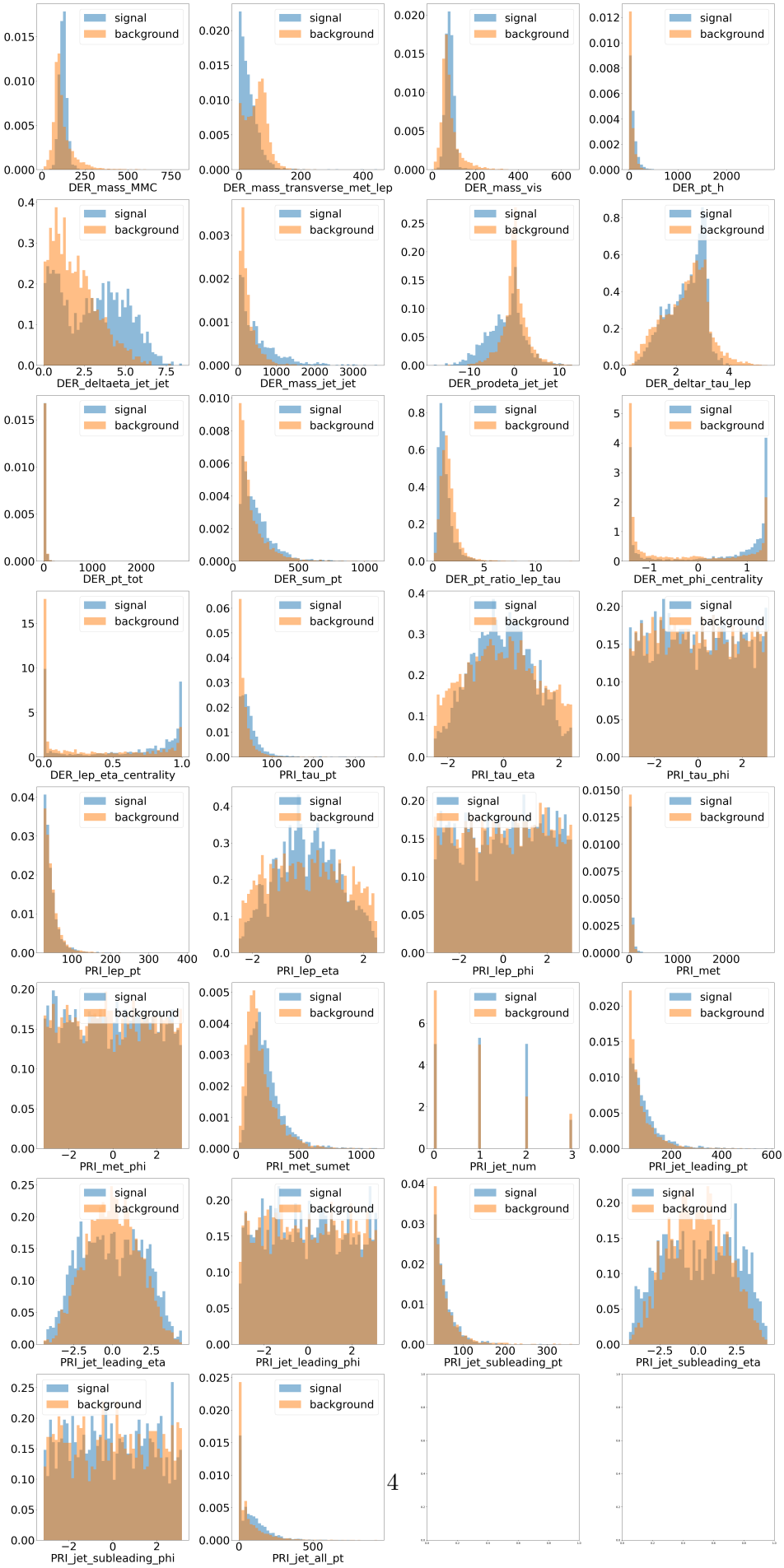
[4]:
```
# look at column labels --- notice last one is "Label" and first is "EventId"␣
 ↪also "Weight"
print(f"Number of signal events: {len(data[data.Label == 's'])}")
print(f"Number of background events: {len(data[data.Label == 'b'])}")
print(f"Fraction signal: {len(data[data.Label == 's'])/(len(data[data.Label ==␣
 ↪'s']) + len(data[data.Label == 'b']))}")
```

```
Number of signal events: 3372
Number of background events: 6628
Fraction signal: 0.3372
```

[5]:
```
plt.figure()


fig, axs = plt.subplots(8, 4, figsize=(40, 80))

for ix, ax in enumerate(axs.reshape(-1)):
    col = data.columns[ix + 1]
    if col == "Weight" or col == "Label":
        continue
    signal = data[col][data.Label == "s"].to_numpy()
    mask_signal = signal > -999
    background = data[col][data.Label == "b"].to_numpy()
    mask_background = background > -999
    xmin = min(np.min(background[mask_background]), np.min(signal[mask_signal]))
    xmax = max(np.max(background[mask_background]), np.max(signal[mask_signal]))

    ax.hist(signal[mask_signal], bins=np.linspace(xmin, xmax, 51), alpha=0.5,␣
 ↪label="signal", density=True)
```

```
    ax.hist(background[mask_background], bins=np.linspace(xmin, xmax, 51),␣
 ↪alpha=0.5, label="background", density=True)

    ax.set_xlabel(col, fontsize=40)
    ax.set_xlabel(col, fontsize=40)
    ax.tick_params(axis="both", which="major", labelsize=40)
    ax.legend(fontsize=40)
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```


signal
background
DER_mass_MMC
DER_mass_transverse_met_lep
DER_mass_vis
DER_pt_h
DER_deltaeta_jet_jet
DER_mass_jet_jet
DER_prodeta_jet_jet
DER_deltar_tau_lep
DER_pt_tot
DER_sum_pt
DER_pt_ratio_lep_tau
DER_met_phi_centrality
DER_lep_eta_centrality
PRI_tau_pt
PRI_tau_eta
PRI_tau_phi
PRI_lep_pt
PRI_lep_eta
PRI_lep_phi
PRI_met
PRI_met_phi
PRI_met_sumet
PRI_jet_num
PRI_jet_leading_pt
PRI_jet_leading_eta
PRI_jet_leading_phi
PRI_jet_subleading_pt
PRI_jet_subleading_eta
PRI_jet_subleading_phi
PRI_jet_all_pt

[6]:
```
data["Label"] = data.Label.astype("category")
data_train = data[:8000]
data_test = data[8000:]
print(f"Number of training samples: {len(data_train)}")
print(f"Number of testing samples: {len(data_test)}")
print()
print(f"Number of signal events in training set: {len(data_train[data_train.
 ↪Label == 's'])}")
print(f"Number of background events in training set: {len(data_train[data_train.
 ↪Label == 'b'])}")
print(
    f"Fraction signal: {len(data_train[data_train.Label == 's'])/
 ↪(len(data_train[data_train.Label == 's']) + len(data_train[data_train.Label␣
 ↪== 'b']))}"
)
```

```
Number of training samples: 8000
Number of testing samples: 2000

Number of signal events in training set: 2688
Number of background events in training set: 5312
Fraction signal: 0.336
```

[7]:
```
feature_names = list(data.columns[1:-2])  # we skip the first and last two␣
 ↪columns because they are the ID, weight, and label

print(len(feature_names))

train = xgb.DMatrix(
    data=data_train[feature_names], label=data_train.Label.cat.codes,␣
 ↪missing=-999.0, feature_names=feature_names
)
test = xgb.DMatrix(
    data=data_test[feature_names], label=data_test.Label.cat.codes,␣
 ↪missing=-999.0, feature_names=feature_names
)
```

```
30
```

[8]:
```
print(f"Number of training samples: {train.num_row()}")
print(f"Number of testing samples: {test.num_row()}")
print()
print(f"Number of signal events in training set: {len(np.where(train.
 ↪get_label())[0])}")
```

```
Number of training samples: 8000
```

```
Number of testing samples: 2000

Number of signal events in training set: 2688
```

[9]:
```
param = {}

param["seed"] = 42  # set seed for reproducibility

# Booster parameters
param["eta"] = 0.1  # learning rate
param["max_depth"] = 10  # maximum depth of a tree
param["subsample"] = 0.8  # fraction of events to train tree on
param["colsample_bytree"] = 0.8  # fraction of features to train tree on

# Learning task parameters
param["objective"] = "binary:logistic"  # objective function
param["eval_metric"] = "error"  # evaluation metric for cross validation, note:␣
 ↪last one is used for early stopping
param = list(param.items())

num_trees = 100  # number of trees to make
```

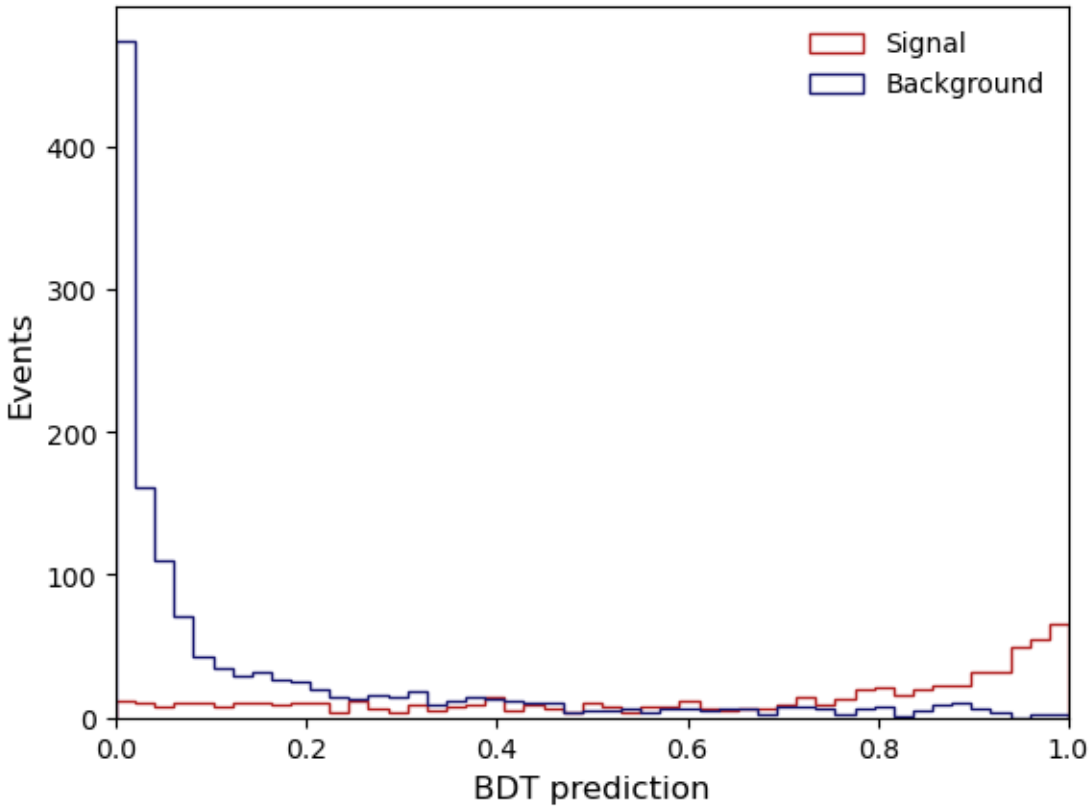
[10]:
```
booster = xgb.train(param, train, num_boost_round=num_trees)
```

[11]:
```
print(booster.eval(test))
```

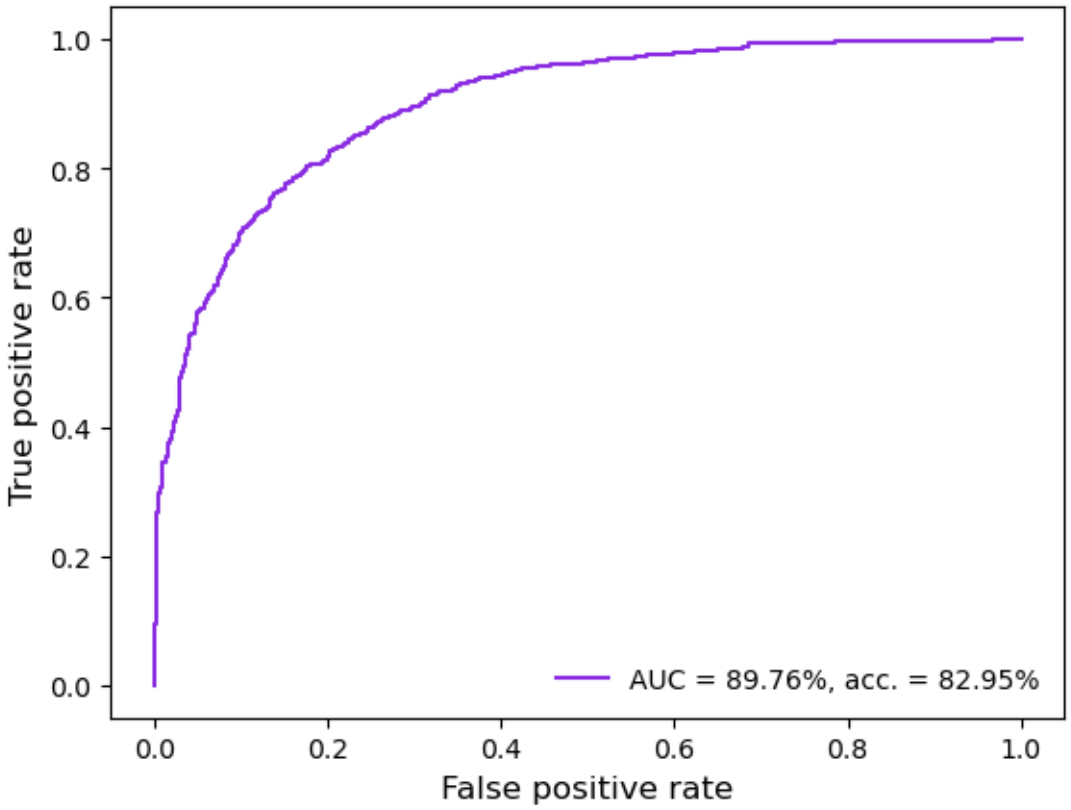
```
[0]     eval-error:0.17050000000000001
```

[12]:
```
predictions = booster.predict(test)
labels = test.get_label().astype(bool)
```

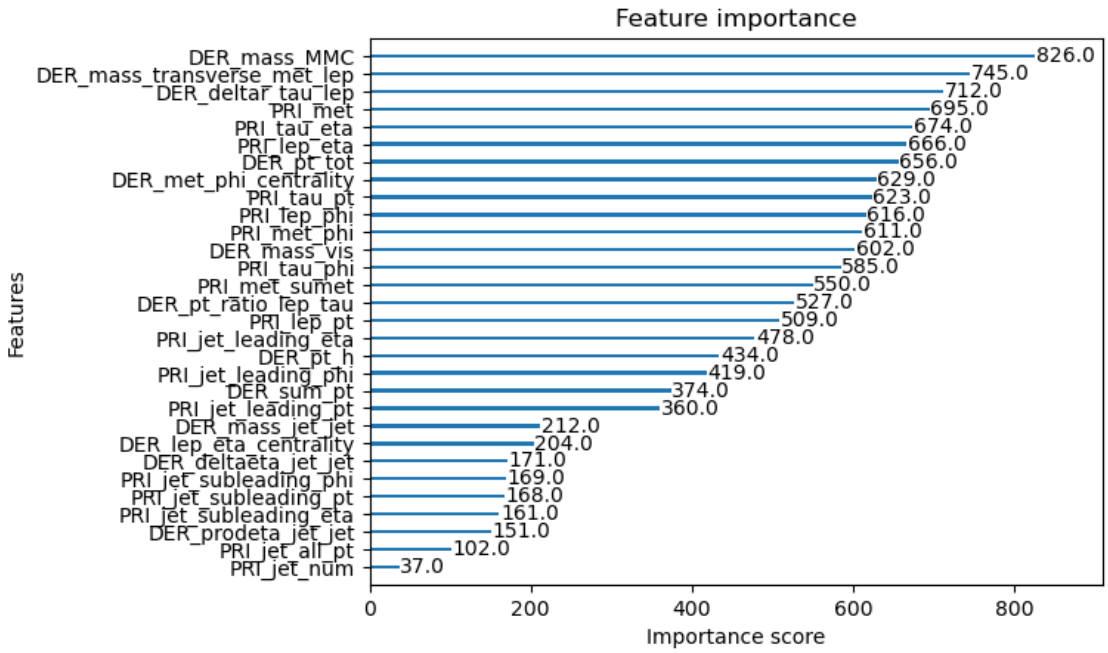
[13]:
```
# plot signal and background predictions, separately
plt.figure()
plt.hist(predictions[labels], bins=np.linspace(0, 1, 50), histtype="step",␣
 ↪color="firebrick", label="Signal")
plt.hist(predictions[~labels], bins=np.linspace(0, 1, 50), histtype="step",␣
 ↪color="midnightblue", label="Background")
# make the plot readable
plt.xlabel("BDT prediction", fontsize=12)
plt.ylabel("Events", fontsize=12)
plt.legend(frameon=False)
plt.xlim(0, 1)
plt.show()
```



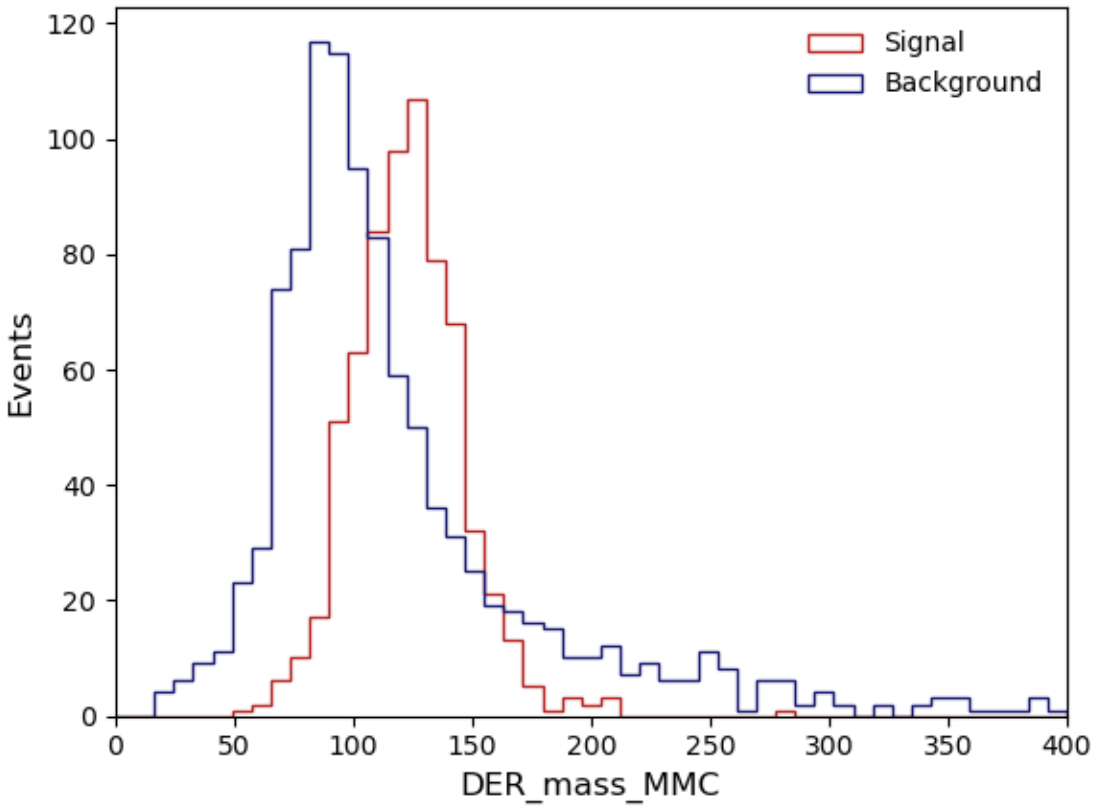

[14]:
```
from sklearn.metrics import roc_curve, auc, accuracy_score

fpr, tpr, _ = roc_curve(labels, predictions)
auc_score = auc(fpr, tpr)
acc_score = accuracy_score(labels, predictions > 0.5)

# plot TPR vs. FPR (ROC curve)
plt.figure()
plt.plot(fpr, tpr, color="blueviolet", label=f"AUC = {auc_score*100:.2f}%, acc.␣
 ↪= {acc_score*100:.2f}%")
# make the plot readable
plt.xlabel("False positive rate", fontsize=12)
plt.ylabel("True positive rate", fontsize=12)
plt.legend(frameon=False)
plt.show()
```




[15]:
```
xgb.plot_importance(booster, grid=False)
plt.show()
```

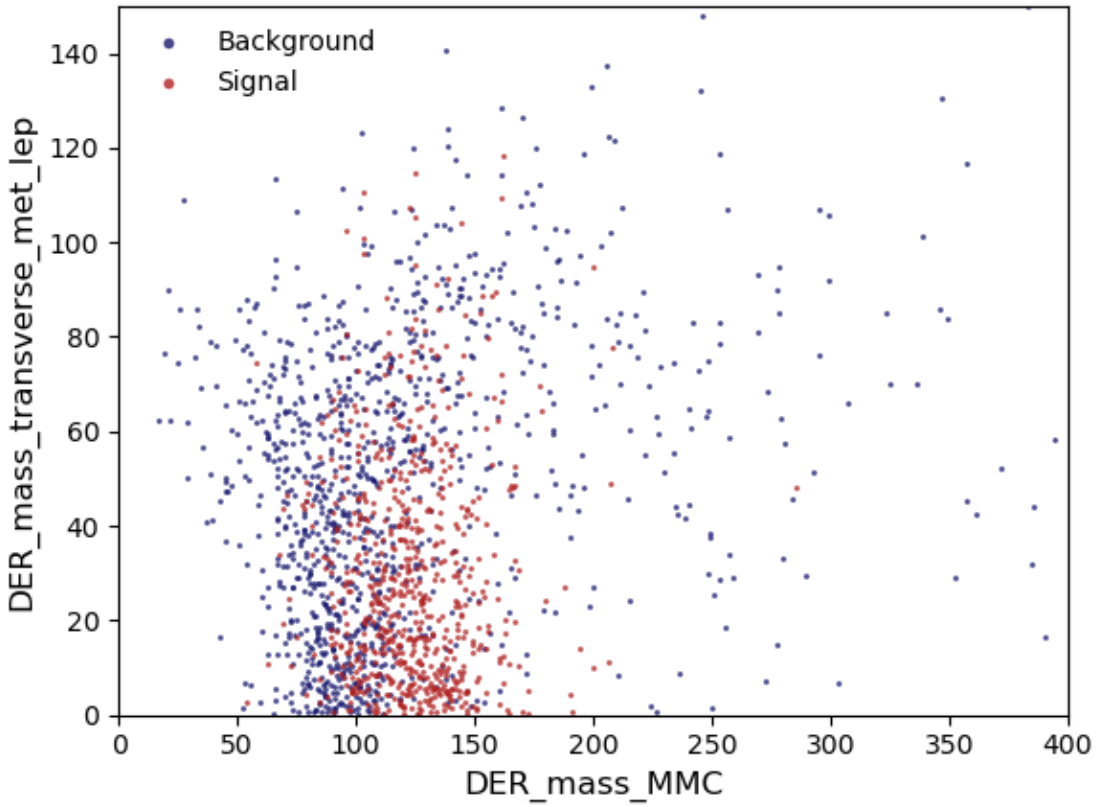



[16]:
```
plt.figure()
plt.hist(
    data_test.DER_mass_MMC[data_test.Label == "s"],
    bins=np.linspace(0, 400, 50),
    histtype="step",
    color="firebrick",
    label="Signal",
)
plt.hist(
    data_test.DER_mass_MMC[data_test.Label == "b"],
    bins=np.linspace(0, 400, 50),
    histtype="step",
    color="midnightblue",
    label="Background",
)

plt.xlim(0, 400)
plt.xlabel("DER_mass_MMC", fontsize=12)
plt.ylabel("Events", fontsize=12)
plt.legend(frameon=False)
plt.show()
```




[17]:
```
plt.figure()

mask_b = np.array(data_test.Label == "b")
mask_s = np.array(data_test.Label == "s")

DER_mass_MMC = np.array(data_test.DER_mass_MMC)
DER_mass_transverse_met_lep = np.array(data_test.DER_mass_transverse_met_lep)

plt.plot(
    DER_mass_MMC[mask_b],
    DER_mass_transverse_met_lep[mask_b],
    "o",
    markersize=2,
    color="midnightblue",
    markeredgewidth=0,
    alpha=0.8,
    label="Background",
)
plt.plot(
    DER_mass_MMC[mask_s],
    DER_mass_transverse_met_lep[mask_s],
```

```
    "o",
    markersize=2,
    color="firebrick",
    markeredgewidth=0,
    alpha=0.8,
    label="Signal",
)
plt.xlim(0, 400)
plt.ylim(0, 150)
plt.xlabel("DER_mass_MMC", fontsize=12)
plt.ylabel("DER_mass_transverse_met_lep", fontsize=12)
plt.legend(frameon=False, numpoints=1, markerscale=2)
plt.show()
```

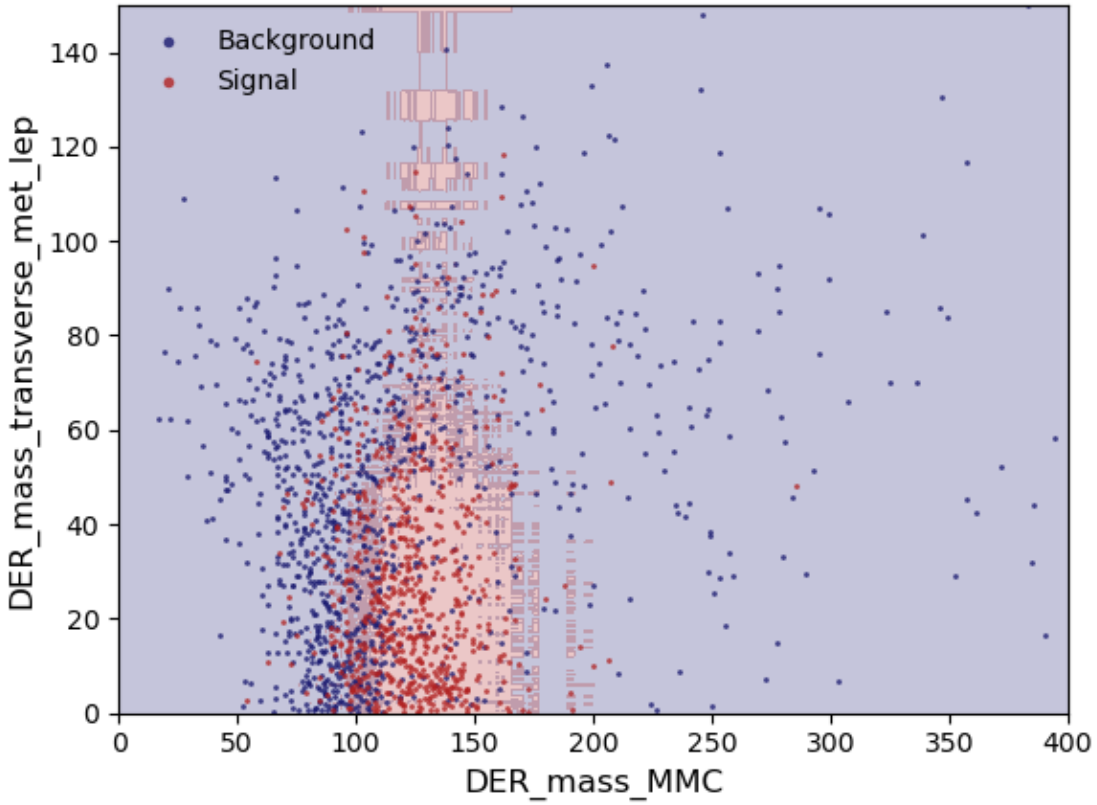



[18]:
```
feature_names = ["DER_mass_MMC", "DER_mass_transverse_met_lep"]

train = xgb.DMatrix(
    data=data_train[feature_names], label=data_train.Label.cat.codes,␣
 ↪missing=-999.0, feature_names=feature_names
)
```

```
test = xgb.DMatrix(
    data=data_test[feature_names], label=data_test.Label.cat.codes,␣
 ↪missing=-999.0, feature_names=feature_names
)

booster = xgb.train(param, train, num_boost_round=num_trees)
```

[19]:
```
from matplotlib.colors import ListedColormap

# first get a mesh grid
x_grid, y_grid = np.meshgrid(np.linspace(0, 400, 1000), np.linspace(0, 150,␣
 ↪1000))
# convert grid into DMatrix
matrix_grid = xgb.DMatrix(
    data=np.c_[x_grid.ravel(), y_grid.ravel()], missing=-999.0,␣
 ↪feature_names=feature_names
)
# run prediction for every value in grid
z_grid = booster.predict(matrix_grid)
# reshape
z_grid = z_grid.reshape(x_grid.shape)


plt.figure()
# plot decision boundary
ax = plt.subplot(111)
cm = ListedColormap(["midnightblue", "firebrick"])
plt.contourf(x_grid, y_grid, z_grid, levels=[0, 0.5, 1], cmap=cm, alpha=0.25)
# overlaid with test data points
plt.plot(
    DER_mass_MMC[mask_b],
    DER_mass_transverse_met_lep[mask_b],
    "o",
    markersize=2,
    color="midnightblue",
    markeredgewidth=0,
    alpha=0.8,
    label="Background",
)
plt.plot(
    DER_mass_MMC[mask_s],
    DER_mass_transverse_met_lep[mask_s],
    "o",
    markersize=2,
    color="firebrick",
    markeredgewidth=0,
```

```
    alpha=0.8,
    label="Signal",
)
ax.set_ylim(0,150)
ax.set_xlim(0,400)
plt.xlabel("DER_mass_MMC", fontsize=12)
plt.ylabel("DER_mass_transverse_met_lep", fontsize=12)
plt.legend(frameon=False, numpoints=1, markerscale=2)
plt.show()
```




What do you notice about the shape of the decision boundary?

The boundary is made of

Do you see any evidence of overfitting? How can you prove it? (Hint: consider the training data)