In [1]:

```
import numpy as np
import pandas as pd
# Loading datasets
train_df = pd.read_csv("training.csv")
test_df = pd.read_csv("test.csv")
agree_df = pd.read_csv("check_agreement.csv")
corr_df = pd.read_csv("check_correlation.csv")
```

writing functions for agreement and correlation test

In [2]:

```
# Check agrrement test


# agreement test as mentioned in kaggle resources#
from sklearn.metrics import roc_curve, auc

def __roc_curve_splitted(data_zero, data_one, sample_weights_zero, sample_we
    """
    Compute roc curve

    :param data_zero: 0-labeled data
    :param data_one:  1-labeled data
    :param sample_weights_zero: weights for 0-labeled data
    :param sample_weights_one:  weights for 1-labeled data
    :return: roc curve
    """
    labels = [0] * len(data_zero) + [1] * len(data_one)
    weights = np.concatenate([sample_weights_zero, sample_weights_one])
    data_all = np.concatenate([data_zero, data_one])
    fpr, tpr, _ = roc_curve(labels, data_all, sample_weight=weights)
    return fpr, tpr

def compute_ks(data_prediction, mc_prediction, weights_data, weights_mc):
    """
    Compute Kolmogorov-Smirnov (ks) distance between real data predictions c

    :param data_prediction: array-like, real data predictions
    :param mc_prediction: array-like, Monte Carlo data predictions
    :param weights_data: array-like, real data weights
    :param weights_mc: array-like, Monte Carlo weights
    :return: ks value
    """
    assert len(data_prediction) == len(weights_data), 'Data length and weigh
    assert len(mc_prediction) == len(weights_mc), 'Data length and weight or

    data_prediction, mc_prediction = np.array(data_prediction), np.array(mc_
    weights_data, weights_mc = np.array(weights_data), np.array(weights_mc)

    assert np.all(data_prediction >= 0.) and np.all(data_prediction <= 1.),
    assert np.all(mc_prediction >= 0.) and np.all(mc_prediction <= 1.), 'MC

    weights_data /= np.sum(weights_data)
    weights_mc /= np.sum(weights_mc)

    fpr, tpr = __roc_curve_splitted(data_prediction, mc_prediction, weights_

    Dnm = np.max(np.abs(fpr - tpr))
    return Dnm

# check correlation test

# correlation test as mentioned in kaggle resources

def __rolling_window(data, window_size):
    """
    Rolling window: take window with definite size through the array
```

```

    :param data: array-like
    :param window_size: size
    :return: the sequence of windows

    Example: data = array(1, 2, 3, 4, 5, 6), window_size = 4
        Then this function return array(array(1, 2, 3, 4), array(2, 3, 4, 5)
    """
    shape = data.shape[:-1] + (data.shape[-1] - window_size + 1, window_size
    strides = data.strides + (data.strides[-1],)
    return np.lib.stride_tricks.as_strided(data, shape=shape, strides=stride

def __cvm(subindices, total_events):
    """
    Compute Cramer-von Mises metric.
    Compared two distributions, where first is subset of second one.
    Assuming that second is ordered by ascending

    :param subindices: indices of events which will be associated with the f
    :param total_events: count of events in the second distribution
    :return: cvm metric
    """
    target_distribution = np.arange(1, total_events + 1, dtype='float') / to
    subarray_distribution = np.cumsum(np.bincount(subindices, minlength=tota
    subarray_distribution /= 1.0 * subarray_distribution[-1]
    return np.mean((target_distribution - subarray_distribution) ** 2)

def compute_cvm(predictions, masses, n_neighbours=200, step=50):
    """
    Computing Cramer-von Mises (cvm) metric on background events: take avera
    In each mass bin global prediction's cdf is compared to prediction's cdf

    :param predictions: array-like, predictions
    :param masses: array-like, in case of Kaggle tau23mu this is reconstruct
    :param n_neighbours: count of neighbours for event to define mass bin
    :param step: step through sorted mass-array to define next center of bin
    :return: average cvm value
    """
    predictions = np.array(predictions)
    masses = np.array(masses)
    assert len(predictions) == len(masses)

    # First, reorder by masses
    predictions = predictions[np.argsort(masses)]

    # Second, replace probabilities with order of probability among other ev
    predictions = np.argsort(np.argsort(predictions, kind='mergesort'), kind

    # Now, each window forms a group, and we can compute contribution of eac
    cvms = []
    for window in __rolling_window(predictions, window_size=n_neighbours)[::
        cvms.append(__cvm(subindices=window, total_events=len(predictions)))
    return np.mean(cvms)
```

function to add new features

In [3]:
```
# feature engineering
def new_feats(df):
    df2 = df.copy()
    df2['isolation_abc'] = df['isolationa'] + df['isolationb'] + df['isolati
    df2['isolation_def'] = df['isolationd'] + df['isolatione'] + df['isolati
    df2['p_IP'] = df['p0_IP']+df['p1_IP']+df['p2_IP']
    df2['p_p']  = df['p0_p']+df['p1_p']+df['p2_p']
    df2['IP_pp'] = df['IP_p0p2'] + df['IP_p1p2']
    df2['p_IPSig'] = df['p0_IPSig'] + df['p1_IPSig'] + df['p2_IPSig']
    #new feature using 'FlightDistance' and LifeTime(from literature)
    df2['FD_LT']=df['FlightDistance']/df['LifeTime']
    #new feature using 'FlightDistance', 'po_p', 'p1_p', 'p2_p'(from literat
    df2['FD_p0p1p2_p']=df['FlightDistance']/(df['p0_p']+df['p1_p']+df['p2_p'
    #new feature using 'LifeTime', 'p0_IP', 'p1_IP', 'p2_IP'(from literature
    df2['NEW5_lt']=df['LifeTime']*(df['p0_IP']+df['p1_IP']+df['p2_IP'])/3
    #new feature using 'p0_track_Chi2Dof', 'p1_track_Chi2Dof', 'p2_track_Chi
    df2['Chi2Dof_MAX'] = df.loc[:, ['p0_track_Chi2Dof', 'p1_track_Chi2Dof',
    # features from kaggle discussion forum
    df2['flight_dist_sig2'] = (df['FlightDistance']/df['FlightDistanceError'
    df2['flight_dist_sig'] = df['FlightDistance']/df['FlightDistanceError']
    df2['NEW_IP_dira'] = df['IP']*df['dira']
    df2['p0p2_ip_ratio']=df['IP']/df['IP_p0p2']
    df2['p1p2_ip_ratio']=df['IP']/df['IP_p1p2']
    df2['DCA_MAX'] = df.loc[:, ['DOCAone', 'DOCAtwo', 'DOCAthree']].max(axis
    df2['iso_bdt_min'] = df.loc[:, ['p0_IsoBDT', 'p1_IsoBDT', 'p2_IsoBDT']].
    df2['iso_min'] = df.loc[:, ['isolationa', 'isolationb', 'isolationc','is
    return df2

```

In [4]:
```
# adding engineered features to training and test datasets
train_df_1 = new_feats(train_df)
test_df_1 = new_feats(test_df)
```

In [5]:
```
# idenifying some features to remove which have been used to engineer new fe
#low importance in EDA
remove = ['id', 'min_ANNmuon', 'production', 'mass', 'signal','SPDhits','CDF
          'p0_pt', 'p1_pt', 'p2_pt','p0_p', 'p1_p', 'p2_p', 'p0_eta', 'p1_et
          'isolationc', 'isolationd', 'isolatione', 'isolationf','p0_IsoBDT'
          'p2_IP','IP_p0p2', 'IP_p1p2','p0_track_Chi2Dof', 'p1_track_Chi2Dof
          'p2_IPSig','DOCAone', 'DOCAtwo', 'DOCAthree']
# making a list of features to be used to train the model and make predictio
features = list(f for f in train_df_1.columns if f not in remove)
```

In [6]:
```
len(features)
```

Out[6]: 28

Creating a new class for UGradientBoosting with loss incorporated in the class itself for bayesian optimization hyper-parameter tuning

In [7]:

```
from hep_ml.gradientboosting import UGradientBoostingClassifier
from hep_ml.losses import BinFlatnessLossFunction
from sklearn.base import BaseEstimator
from sklearn.metrics import accuracy_score
from collections import Counter
class UGradientBoostingClassifierWithLoss(BaseEstimator):
    def __init__(
#           self, max_depth=3, max_features=0.8, learning_rate=0.01,
#           n_estimators=80, subsample=0.8
        self, max_depth, n_estimators, **params
    ):
        loss = BinFlatnessLossFunction(
            ['mass'], n_bins=15, uniform_label = 0 , fl_coefficient=15, powe
        )

        self.estimator = UGradientBoostingClassifier(
            loss=loss,
            train_features = list(f for f in train_df_1.columns if f not in
            max_depth = max_depth,
            n_estimators = n_estimators,
            **params
        )

    def fit(self, X, y=None):
        self.estimator.fit(X, y)

        return self

    def predict_proba(self, X):
        return self.estimator.predict_proba(X)

    def predict(self, X):
        return self.estimator.predict(X)

    def transform(self, X):
        return self.estimator.transform(X)

    def get_params(self, deep=True):
        # suppose this estimator has parameters "alpha" and "recursive"
        params_to_keep = [
             "max_depth",
            "max_features",
            "learning_rate",
             "n_estimators",
            "subsample",
        ]

        ret = dict()
        tret = self.estimator.get_params(deep=deep)
        for key in params_to_keep:
            ret[key] = tret[key]

        return ret

    def set_params(self, **parameters):
        self.estimator.get_params(parameters)
```

```
        return self

    def score(self, X, y):
        y_pred = self.estimator.predict(X)

        acc = accuracy_score(y, y_pred)
        print(acc)
        print(Counter(y))
        print(Counter(y_pred))

        return acc

```

In [8]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold

from hep_ml.gradientboosting import UGradientBoostingClassifier
from hep_ml.losses import BinFlatnessLossFunction
from bayes_opt import BayesianOptimization
from sklearn.metrics import roc_auc_score, make_scorer, accuracy_score
import time
import json
auc = make_scorer(roc_auc_score)
```

bayesian optimization demands bounds of hyperparameters. Therefore parameters with discrete values can not be tuned with this technique. The discrete hyperparameters neede to be tuned are n_estimators and max_depth. Therefore creating all possible pairs of these two parameters and doing grid search for these two parameters while running bayesian optimization for each pair of these two discrete parameters

In [23]:

```python
start = time.time()
# we will save best parameters in best dictionary
best = dict()
best['score'] = 0
import itertools
max_depth = [3,6,9]
n_estimators = [50,400,900]
for x in itertools.product(max_depth, n_estimators):

    print("max_depth-n_estimator pair: {}".format(x))
    def gbm_cl_bo(max_features, learning_rate, subsample):

        params_gbm = {}
    #   params_gbm['max_depth'] = round(max_depth)
        params_gbm['max_features'] = max_features
        params_gbm['learning_rate'] = learning_rate
    #   params_gbm['n_estimators'] = round(n_estimators)
        params_gbm['subsample'] = subsample


    #     loss = BinFlatnessLossFunction(['mass'], n_bins=15, uniform_label
        skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
        skf.get_n_splits(train_df_1[features + ['mass']], train_df_1['signal

        scores = cross_val_score(UGradientBoostingClassifierWithLoss(max_dep
                                 train_df_1[features + ['mass']], train_df_1
                                 scoring = auc,
                                 cv=skf)
        score = scores.mean()
        #print("max_depth: {}, n_estimators: {}, max_features: {}, learning_
        #      .format(x[0], x[1], params_gbm['max_features'], params_gbm['l

        if score > best['score']:
            best['score'] = score
            best['max_depth'] = x[0]
            best['n_estimators'] = x[1]
            best['max_features'] = params_gbm['max_features']
            best['learning_rate'] = params_gbm['learning_rate']
            best['subsample'] = params_gbm['subsample']

        return score

    params_gbm ={
#    'max_depth': (3),
    'max_features':(0.5, 1),
    'learning_rate':(0.01, 1),
#    'n_estimators': (100),
    'subsample': (0.5, 1)
    }

    gbm_bo = BayesianOptimization(gbm_cl_bo, params_gbm, random_state=111)
    gbm_bo.maximize(init_points=10, n_iter=3)
```

```
57 print('It takes %s minutes' % ((time.time() - start)/60))
```

```
=============================================================
max_depth-n_estimator pair: (9, 900)
|   iter    |  target   | learni... | max_fe... | subsample |
-------------------------------------------------------------
|  1        |  0.8736   |  0.616    |  0.5845   |  0.718    |
|  2        |  0.8736   |  0.7716   |  0.6477   |  0.5746   |
|  3        |  0.8706   |  0.03225  |  0.7101   |  0.6193   |
|  4        |  0.8811   |  0.3443   |  0.9954   |  0.6189   |
|  5        |  0.876    |  0.09038  |  0.8348   |  0.8106   |
|  6        |  0.8793   |  0.2815   |  0.7331   |  0.5592   |
|  7        |  0.8758   |  0.08322  |  0.9504   |  0.897    |
|  8        |  0.8681   |  0.8422   |  0.9076   |  0.9955   |
|  9        |  0.8762   |  0.5815   |  0.9069   |  0.7107   |
|  10       |  0.8722   |  0.03717  |  0.7271   |  0.5527   |
|  11       |  0.8818   |  0.4855   |  0.9904   |  0.504    |
|  12       |  0.877    |  0.0682   |  0.999    |  0.5176   |
|  13       |  0.8817   |  0.4756   |  0.9829   |  0.5015   |
=============================================================
It takes 1945.4509294231732 minutes
```

took nearly 35 hours to run!!

max_depth-n_estimator pair: (6, 900)

iter | target | learni... | max_fe... | subsample |

12 | 0.8887 | 0.4592 | 0.9965 | 0.5016 |

4 | 0.8866 | 0.3443 | 0.9954 | 0.6189

13 | 0.8867 | 0.3242 | 0.9863 | 0.5006

2 | 0.881 | 0.7716 | 0.6477 | 0.5746

In [24]:
```
best
```

Out[24]:
```
{'score': 0.8886783798768416,
 'max_depth': 6,
 'n_estimators': 900,
 'max_features': 0.9965346255608354,
 'learning_rate': 0.459200553861577,
 'subsample': 0.501638892280019}
```

it is found that the hyperparameter which gives best score does not pass correlation test. Therefore trying other hyperparameters with best scores in descending order

In [16]:
```
UGradientBoostingClassifier?
```

following model with passed parameters passes both the required tests. Therefore using the results from this model to check on kaggle test data

In [17]:

```
loss = BinFlatnessLossFunction(['mass'], n_bins=15, uniform_label=0 , fl_coe
model = UGradientBoostingClassifier(loss=loss, n_estimators=900,
                                    max_depth = 6,
                                    learning_rate = 0.1,
                                    train_features = features,
                                    subsample=0.5)
model.fit(train_df_1[features + ['mass']], train_df_1['signal'])
```

Out[17]:

```
UGradientBoostingClassifier(learning_rate=0.1,
                            loss=BinFlatnessLossFunction(allow_wrong_signs=Tru
e,
                                                         fl_coefficient=15,
                                                         n_bins=15, power=2,
                                                         uniform_features=['mas
s'],
                                                         uniform_label=array
([0])),
                            max_depth=6, max_features=None, max_leaf_nodes=Non
e,
                            min_samples_leaf=1, min_samples_split=2,
                            n_estimators=900,
                            random_state=RandomState(MT19937) at 0x290F6367740,
                            splitter=...
                            train_features=['LifeTime', 'dira',
                                            'FlightDistance',
                                            'FlightDistanceError', 'IP',
                                            'IPSig', 'VertexChi2', 'pt', 'iso',
                                            'ISO_SumBDT', 'isolation_abc',
                                            'isolation_def', 'p_IP', 'p_p',
                                            'IP_pp', 'p_IPSig', 'FD_LT',
                                            'FD_p0p1p2_p', 'NEW5_lt',
                                            'Chi2Dof_MAX', 'flight_dist_sig2',
                                            'flight_dist_sig', 'NEW_IP_dira',
                                            'p0p2_ip_ratio', 'p1p2_ip_ratio',
                                            'DCA_MAX', 'iso_bdt_min',
                                            'iso_min'],
                            update_tree=True)
```

In [22]:

```
# saving the model to the memory
import pickle
filename = 'finalized_model_1.sav'
pickle.dump(model, open(filename, 'wb'))
```

In [23]:

```
# Loading the model from memory
loaded_model = pickle.load(open(filename, 'rb'))
```

In [24]:

```
loaded_model
```

Out[24]:

```
UGradientBoostingClassifier(learning_rate=0.1,
                            loss=BinFlatnessLossFunction(allow_wrong_signs=Tru
e,
                                                         fl_coefficient=15,
                                                         n_bins=15, power=2,
                                                         uniform_features=['mas
s'],
                                                         uniform_label=array
([0])),
                            max_depth=6, max_features=None, max_leaf_nodes=Non
e,
                            min_samples_leaf=1, min_samples_split=2,
                            n_estimators=900,
                            random_state=RandomState(MT19937) at 0x290A3B58140,
                            splitter=...
                            train_features=['LifeTime', 'dira',
                                            'FlightDistance',
                                            'FlightDistanceError', 'IP',
                                            'IPSig', 'VertexChi2', 'pt', 'iso',
                                            'ISO_SumBDT', 'isolation_abc',
                                            'isolation_def', 'p_IP', 'p_p',
                                            'IP_pp', 'p_IPSig', 'FD_LT',
                                            'FD_p0p1p2_p', 'NEW5_lt',
                                            'Chi2Dof_MAX', 'flight_dist_sig2',
                                            'flight_dist_sig', 'NEW_IP_dira',
                                            'p0p2_ip_ratio', 'p1p2_ip_ratio',
                                            'DCA_MAX', 'iso_bdt_min',
                                            'iso_min'],
                            update_tree=True)
```

In [19]:

```
# conducting agreement check test
check_agreement = pd.read_csv("check_agreement.csv")
check_agreement = new_feats(check_agreement)

#check_agreement = pandas.read_csv(folder + 'check_agreement.csv', index_col
agreement_probs = model.predict_proba(check_agreement[features])[:, 1]

ks = compute_ks(
    agreement_probs[check_agreement['signal'].values == 0],
    agreement_probs[check_agreement['signal'].values == 1],
    check_agreement[check_agreement['signal'] == 0]['weight'].values,
    check_agreement[check_agreement['signal'] == 1]['weight'].values)
#print 'KS metric', ks, ks < 0.09
print("KS metric {}".format(ks))
print(ks < 0.09)
```

```
KS metric 0.0795798778412874
True
```

In [18]:

```
# conducting the correlation test
#check_correlation = pandas.read_csv(folder + 'check_correlation.csv', index_
check_correlation = pd.read_csv("check_correlation.csv", index_col = "id")
check_correlation = new_feats(check_correlation)
correlation_probs = model.predict_proba(check_correlation[features])[:, 1]
cvm = compute_cvm(correlation_probs, check_correlation['mass'])
#print 'CvM metric', cvm, cvm < 0.002
print("CvM metric {}".format(cvm))
print(cvm < 0.002)
```

```
CvM metric 0.00135303046269074
True
```

In [21]:

```
# making submission file with test dataset to be submitted on kaggle
test_probs = model.predict_proba(test_df_1[features])[:,1]
result = pd.DataFrame({"id": test_df["id"], "prediction": test_probs})
result.to_csv("final_result_1.csv", index=False)
```

20th rank with final_result_1.csv

In [ ]:

```

```