In [1]:
```
# Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# 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")
```

In [3]:
```
from hep_ml.gradientboosting import UGradientBoostingClassifier
from hep_ml.losses import BinFlatnessLossFunction
```

In [4]:
```
df2 = train_df.copy()
```

In [5]:
```
df2.shape
```

Out[5]:
```
(67553, 51)
```

In [6]:
```
train_df.shape
```

Out[6]:
```
(67553, 51)
```

training UGradientBoosting model with train dataset with columns which are in test dataset

In [7]:
```
remove = ['id', 'min_ANNmuon', 'production', 'mass', 'signal']
features = list(f for f in train_df.columns if f not in remove)
```

In [8]:
```
print("Train a UGradientBoostingClassifier")
loss = BinFlatnessLossFunction(['mass'], n_bins=15, uniform_label=0 , fl_coeffici
model = UGradientBoostingClassifier(loss=loss, n_estimators=900,
                                    max_depth=6,
                                    learning_rate=0.15,
                                    train_features=features,
                                    subsample=0.7)
model.fit(train_df[features + ['mass']], train_df['signal'])
```

```
Train a UGradientBoostingClassifier
```

Out[8]:
```
UGradientBoostingClassifier(learning_rate=0.15,
                            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 0x1AEE92DC940,
                            splitter...
                                        'FlightDistance',
                                        'FlightDistanceError', 'IP',
                                        'IPSig', 'VertexChi2', 'pt',
                                        'DOCAone', 'DOCAtwo', 'DOCAthree',
                                        'IP_p0p2', 'IP_p1p2', 'isolationa',
                                        'isolationb', 'isolationc',
                                        'isolationd', 'isolatione',
                                        'isolationf', 'iso', 'CDF1', 'CDF
2',
                                        'CDF3', 'ISO_SumBDT', 'p0_IsoBDT',
                                        'p1_IsoBDT', 'p2_IsoBDT',
                                        'p0_track_Chi2Dof',
                                        'p1_track_Chi2Dof',
                                        'p2_track_Chi2Dof', ...],
                            update_tree=True)
```

Check agrrement test

In [11]:
```
from sklearn.metrics import roc_curve, auc

def __roc_curve_splitted(data_zero, data_one, sample_weights_zero, sample_weights
    """
    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 cdf ar

    :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 weight one
    assert len(mc_prediction) == len(weights_mc), 'Data length and weight one mus

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

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

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

    fpr, tpr = __roc_curve_splitted(data_prediction, mc_prediction, weights_data,

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

check correlation test

In [12]:
```
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), arr
    """
    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=strides)

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 first
    :param total_events: count of events in the second distribution
    :return: cvm metric
    """
    target_distribution = np.arange(1, total_events + 1, dtype='float') / total_e
    subarray_distribution = np.cumsum(np.bincount(subindices, minlength=total_eve
    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 average of
    In each mass bin global prediction's cdf is compared to prediction's cdf in m

    :param predictions: array-like, predictions
    :param masses: array-like, in case of Kaggle tau23mu this is reconstructed ma
    :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 events
    predictions = np.argsort(np.argsort(predictions, kind='mergesort'), kind='mer

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

In [ ]:

In [13]:
```
check_agreement = pd.read_csv("check_agreement.csv")
#check_agreement = pandas.read_csv(folder + 'check_agreement.csv', index_col='id
agreement_probs = model.predict_proba(check_agreement.drop(['signal', 'id', 'weig

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 0.21436157140235346
False
```

In [ ]:

In [14]:
```
#check_correlation = pandas.read_csv(folder + 'check_correlation.csv', index_col=
check_correlation = pd.read_csv("check_correlation.csv", index_col = "id")
correlation_probs = model.predict_proba(check_correlation.drop(['mass'], axis = 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 0.0009934509874436174
True
```

the agreement test fails with this model. So let us create new features and retrain our model

In [15]:
```
train_df.columns
```

Out[15]:
```
Index(['id', 'LifeTime', 'dira', 'FlightDistance', 'FlightDistanceError', 'IP',
       'IPSig', 'VertexChi2', 'pt', 'DOCAone', 'DOCAtwo', 'DOCAthree',
       'IP_p0p2', 'IP_p1p2', 'isolationa', 'isolationb', 'isolationc',
       'isolationd', 'isolatione', 'isolationf', 'iso', 'CDF1', 'CDF2', 'CDF3',
       'ISO_SumBDT', 'p0_IsoBDT', 'p1_IsoBDT', 'p2_IsoBDT', 'p0_track_Chi2Dof',
       'p1_track_Chi2Dof', 'p2_track_Chi2Dof', 'p0_IP', 'p1_IP', 'p2_IP',
       'p0_IPSig', 'p1_IPSig', 'p2_IPSig', 'p0_pt', 'p1_pt', 'p2_pt', 'p0_p',
       'p1_p', 'p2_p', 'p0_eta', 'p1_eta', 'p2_eta', 'SPDhits', 'production',
       'signal', 'mass', 'min_ANNmuon'],
      dtype='object')
```

In [16]:
```
#here are some features like 'isolationa', 'isolationb', 'isolationc' and 'p0_IP
#their name sound like are related and represent same physical quantity in some w
#create new features
def new_feats(df):
    df2 = df.copy()
    df2['isolation_abc'] = df['isolationa'] + df['isolationb'] + df['isolationc']
    df2['isolation_def'] = df['isolationd'] + df['isolatione'] + df['isolationf']
    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']
    return df2
```

In [17]:
```
train_df_1 = new_feats(train_df)
test_df_1 = new_feats(test_df)
```

In [18]:
```
train_df.shape
```

Out[18]: (67553, 51)

In [19]:
```
train_df_1.shape
```

Out[19]: (67553, 57)

Now let us retrain the model with these new features

In [20]:
```
remove = ['id', 'min_ANNmuon', 'production', 'mass', 'signal']
features = list(f for f in train_df.columns if f not in remove)
print("Train a UGradientBoostingClassifier with new features")
loss = BinFlatnessLossFunction(['mass'], n_bins=15, uniform_label=0 , fl_coeffici
model = UGradientBoostingClassifier(loss=loss, n_estimators=900,
                                    max_depth=6,
                                    learning_rate=0.15,
                                    train_features=features,
                                    subsample=0.7)
model.fit(train_df_1[features + ['mass']], train_df_1['signal'])
```

```
Train a UGradientBoostingClassifier with new features
```

Out[20]:
```
UGradientBoostingClassifier(learning_rate=0.15,
                            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 0x1AEE92DC940,
                            splitter...
                                          'FlightDistance',
                                          'FlightDistanceError', 'IP',
                                          'IPSig', 'VertexChi2', 'pt',
                                          'DOCAone', 'DOCAtwo', 'DOCAthree',
                                          'IP_p0p2', 'IP_p1p2', 'isolationa',
                                          'isolationb', 'isolationc',
                                          'isolationd', 'isolatione',
                                          'isolationf', 'iso', 'CDF1', 'CDF
2',
                                          'CDF3', 'ISO_SumBDT', 'p0_IsoBDT',
                                          'p1_IsoBDT', 'p2_IsoBDT',
                                          'p0_track_Chi2Dof',
                                          'p1_track_Chi2Dof',
                                          'p2_track_Chi2Dof', ...],
                            update_tree=True)
```

In [21]:
```
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='id
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 0.23424866937161137
False
```

In [22]:
```
#check_correlation = pandas.read_csv(folder + 'check_correlation.csv', index_col=
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 0.001074948150581736
True
```

let us try removing some unimportant features based on EDA

In [23]:
```
remove = ['id', 'min_ANNmuon', 'production', 'mass', 'signal','p2_eta','p0_p','is
          'p0_eta','isolationa', 'isolationb', 'isolationc', 'isolationd','isolat
          'CDF1','p0_pt','p1_pt', 'p2_pt','p1_IP','p2_p','FlightDistance','p1_tra
          'FlightDistanceError','p0_IP','p1_IPSig','p2_track_Chi2Dof','pt', 'p0_I
          'p2_p', 'p0_IPSig', 'p1_IPSig', 'p2_IPSig']
```

In [24]:
```
features = list(f for f in train_df.columns if f not in remove)
```

In [25]:
```
train_df_1 = new_feats(train_df)
test_df_1 = new_feats(test_df)
```

In [28]:
```
print("Train a UGradientBoostingClassifier with new features and removing some fe
loss = BinFlatnessLossFunction(['mass'], n_bins=15, uniform_label=0 , fl_coeffici
model = UGradientBoostingClassifier(loss=loss, n_estimators=900,
                                    max_depth=6,
                                    learning_rate=0.15,
                                    train_features=features,
                                    subsample=0.7)
model.fit(train_df_1[features + ['mass']], train_df_1['signal'])
```

```
Train a UGradientBoostingClassifier with new features and removing some feature
s
```

Out[28]:
```
UGradientBoostingClassifier(learning_rate=0.15,
                            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 0x1AEE92DC940,
                            splitter='best', subsample=0.7,
                            train_features=['LifeTime', 'dira', 'IP', 'IPSig',
                                            'VertexChi2', 'DOCAone', 'DOCAtwo',
                                            'DOCAthree', 'IP_p0p2', 'IP_p1p2',
                                            'iso', 'CDF2', 'CDF3', 'ISO_SumBD
T',
                                            'p0_IsoBDT', 'p0_track_Chi2Dof',
                                            'SPDhits'],
                            update_tree=True)
```

In [58]:
```
train_df_1[features + ['mass']].shape
```

Out[58]:
```
(67553, 23)
```

In [59]:
```
 train_df_1['signal'].shape
```

Out[59]:
```
(67553,)
```

In [60]:
```
np.ravel(train_df_1['signal']).shape
```

Out[60]:
```
(67553,)
```

In [29]:
```
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='id
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 0.22324408272050666
False
```

In [30]:
```
#check_correlation = pandas.read_csv(folder + 'check_correlation.csv', index_col=
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 0.0010256531459170882
True
```

the agreement test is still failing. Let us add some more features from lierature where similar problem has been solved

In [31]:
```
def new_feats(df):
    df2 = df.copy()
    df2['isolation_abc'] = df['isolationa'] + df['isolationb'] + df['isolationc']
    df2['isolation_def'] = df['isolationd'] + df['isolatione'] + df['isolationf']
    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 'FlightDu=istance' and LifeTime(from literature)
    df2['FD_LT']=df['FlightDistance']/df['LifeTime']
    #new feature using 'FlightDistance', 'po_p', 'p1_p', 'p2_p'(from literature)
    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_Chi2Dof
    df2['Chi2Dof_MAX'] = df.loc[:, ['p0_track_Chi2Dof', 'p1_track_Chi2Dof', 'p2_t
    return df2
```

In [32]:
```
remove = ['id', 'min_ANNmuon', 'production', 'mass', 'signal','p2_eta','p0_p','is
          'p0_eta','isolationa', 'isolationb', 'isolationc', 'isolationd','isolat
          'CDF1','p0_pt','p1_pt', 'p2_pt','p1_IP','p2_p','FlightDistance','p1_tra
          'FlightDistanceError','p0_IP','p1_IPSig','p2_track_Chi2Dof','pt', 'p0_I
          'p2_p', 'p0_IPSig', 'p1_IPSig', 'p2_IPSig']

features = list(f for f in train_df.columns if f not in remove)
```

In [33]:
```
train_df_1 = new_feats(train_df)
test_df_1 = new_feats(test_df)
```

In [34]:
```
print("Train a UGradientBoostingClassifier with new features and removing some fe
loss = BinFlatnessLossFunction(['mass'], n_bins=15, uniform_label=0 , fl_coeffici
model = UGradientBoostingClassifier(loss=loss, n_estimators=900,
                                    max_depth=6,
                                    learning_rate=0.15,
                                    train_features=features,
                                    subsample=0.7)
model.fit(train_df_1[features + ['mass']], train_df_1['signal'])
```

```
Train a UGradientBoostingClassifier with new features and removing some feature
s
```

Out[34]:
```
UGradientBoostingClassifier(learning_rate=0.15,
                            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 0x1AEE92DC940,
                            splitter='best', subsample=0.7,
                            train_features=['LifeTime', 'dira', 'IP', 'IPSig',
                                            'VertexChi2', 'DOCAone', 'DOCAtwo',
                                            'DOCAthree', 'IP_p0p2', 'IP_p1p2',
                                            'iso', 'CDF2', 'CDF3', 'ISO_SumBD
T',
                                            'p0_IsoBDT', 'p0_track_Chi2Dof',
                                            'SPDhits'],
                            update_tree=True)
```

In [35]:
```
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='id
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 0.2191779035101546
False
```

In [36]:
```
#check_correlation = pandas.read_csv(folder + 'check_correlation.csv', index_col=
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 0.001090672320032115
True
```

the agreement test is still failing. Let us try adding some more features

In [52]:
```
def new_feats(df):

    df2 = df.copy()
    df2['isolation_abc'] = df['isolationa'] + df['isolationb'] + df['isolationc']
    df2['isolation_def'] = df['isolationd'] + df['isolatione'] + df['isolationf']
    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 'FlightDu=istance' and LifeTime(from literature)
    df2['FD_LT']=df['FlightDistance']/df['LifeTime']
    #new feature using 'FlightDistance', 'po_p', 'p1_p', 'p2_p'(from literature)
    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_Chi2Dof
    df2['Chi2Dof_MAX'] = df.loc[:, ['p0_track_Chi2Dof', 'p1_track_Chi2Dof', 'p2_t
    return df2
```

In [53]:
```
train_df_1 = new_feats(train_df)
test_df_1 = new_feats(test_df)
```

In [54]:
```
train_df_1.shape
```

Out[54]:
```
(67553, 61)
```

In [55]:
```
# droping some more unimportant features
remove = ['id', 'min_ANNmuon', 'production', 'mass', 'signal','SPDhits','CDF1',
          'p0_pt', 'p1_pt', 'p2_pt','p0_p', 'p1_p', 'p2_p', 'p0_eta', 'p1_eta',
          'isolationc','isolationd', 'isolatione', 'isolationf','p0_IsoBDT', 'p1_
          'p2_IP','IP_p0p2','IP_p1p2','p0_track_Chi2Dof', 'p1_track_Chi2Dof', 'p2
          'p2_IPSig','DOCAone','DOCAtwo', 'DOCAthree']
features = list(f for f in train_df.columns if f not in remove)
```

In [56]:
```
train_df_1.columns
```

Out[56]:
```
Index(['id', 'LifeTime', 'dira', 'FlightDistance', 'FlightDistanceError', 'IP',
       'IPSig', 'VertexChi2', 'pt', 'DOCAone', 'DOCAtwo', 'DOCAthree',
       'IP_p0p2', 'IP_p1p2', 'isolationa', 'isolationb', 'isolationc',
       'isolationd', 'isolatione', 'isolationf', 'iso', 'CDF1', 'CDF2', 'CDF3',
       'ISO_SumBDT', 'p0_IsoBDT', 'p1_IsoBDT', 'p2_IsoBDT', 'p0_track_Chi2Dof',
       'p1_track_Chi2Dof', 'p2_track_Chi2Dof', 'p0_IP', 'p1_IP', 'p2_IP',
       'p0_IPSig', 'p1_IPSig', 'p2_IPSig', 'p0_pt', 'p1_pt', 'p2_pt', 'p0_p',
       'p1_p', 'p2_p', 'p0_eta', 'p1_eta', 'p2_eta', 'SPDhits', 'production',
       'signal', 'mass', 'min_ANNmuon', 'isolation_abc', 'isolation_def',
       'p_IP', 'p_p', 'IP_pp', 'p_IPSig', 'FD_LT', 'FD_p0p1p2_p', 'NEW5_lt',
       'Chi2Dof_MAX'],
      dtype='object')
```

In [57]:
```
print("Train a UGradientBoostingClassifier with new features and removing some fe
loss = BinFlatnessLossFunction(['mass'], n_bins=15, uniform_label=0 , fl_coeffici
model = UGradientBoostingClassifier(loss=loss, n_estimators=900,
                                    max_depth=6,
                                    learning_rate=0.15,
                                    train_features=features,
                                    subsample=0.7)
model.fit(train_df_1[features + ['mass']], train_df_1['signal'])
```

```
Train a UGradientBoostingClassifier with new features and removing some feature
s
```

Out[57]:
```
UGradientBoostingClassifier(learning_rate=0.15,
                            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 0x1AEE92DC940,
                            splitter='best', subsample=0.7,
                            train_features=['LifeTime', 'dira',
                                            'FlightDistance',
                                            'FlightDistanceError', 'IP',
                                            'IPSig', 'VertexChi2', 'pt', 'iso',
                                            'ISO_SumBDT'],
                            update_tree=True)
```

In [58]:
```
train_df_1[features + ['mass']].columns
```

Out[58]:
```
Index(['LifeTime', 'dira', 'FlightDistance', 'FlightDistanceError', 'IP',
       'IPSig', 'VertexChi2', 'pt', 'iso', 'ISO_SumBDT', 'mass'],
      dtype='object')
```

In [59]:
```
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='id
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 0.04482301271830369
True
```

In [60]:
```
#check_correlation = pandas.read_csv(folder + 'check_correlation.csv', index_col=
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 0.0009370011153217489
True
```

In [48]:
```
test_df = new_feats(test_df)
```

In [61]:
```
test_probs = model.predict_proba(test_df[features])[:,1]
result = pd.DataFrame({"id": test_df["id"], "prediction": test_probs})
result.to_csv("result3.csv", index=False)
```

adding some more features from kaggle discussion forums to enhance the performance

In [81]:
```
def new_feats(df):
    df2 = df.copy()
    df2['isolation_abc'] = df['isolationa'] + df['isolationb'] + df['isolationc']
    df2['isolation_def'] = df['isolationd'] + df['isolatione'] + df['isolationf']
    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 'FlightDu=istance' and LifeTime(from literature)
    df2['FD_LT']=df['FlightDistance']/df['LifeTime']
    #new feature using 'FlightDistance', 'po_p', 'p1_p', 'p2_p'(from literature)
    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_Chi2Dof
    df2['Chi2Dof_MAX'] = df.loc[:, ['p0_track_Chi2Dof', 'p1_track_Chi2Dof', 'p2_t
    # features from kaggle discussion forum
    df2['flight_dist_sig2'] = (df['FlightDistance']/df['FlightDistanceError'])**2
    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=1)
    df2['iso_bdt_min'] = df.loc[:, ['p0_IsoBDT', 'p1_IsoBDT', 'p2_IsoBDT']].min(a
    df2['iso_min'] = df.loc[:, ['isolationa', 'isolationb', 'isolationc','isolati
    return df2
```

In [82]:
```
train_df_1 = new_feats(train_df)
test_df_1 = new_feats(test_df)
```

In [86]:
```
remove = ['id', 'min_ANNmuon', 'production', 'mass', 'signal','SPDhits','CDF1',
          'p0_pt', 'p1_pt', 'p2_pt','p0_p', 'p1_p', 'p2_p', 'p0_eta', 'p1_eta',
          'isolationc', 'isolationd', 'isolatione', 'isolationf','p0_IsoBDT', 'p1
          'p2_IP','IP_p0p2', 'IP_p1p2','p0_track_Chi2Dof', 'p1_track_Chi2Dof', 'p
          'p2_IPSig','DOCAone', 'DOCAtwo', 'DOCAthree']
features = list(f for f in train_df_1.columns if f not in remove)
```

In [88]:
```
len(features)
```

Out[88]: 28

In [89]:
```
print("Train a UGradientBoostingClassifier with new features and removing some fe
loss = BinFlatnessLossFunction(['mass'], n_bins=15, uniform_label=0 , fl_coeffici
model = UGradientBoostingClassifier(loss=loss, n_estimators=900,
                                    max_depth=6,
                                    learning_rate=0.15,
                                    train_features=features,
                                    subsample=0.7)
model.fit(train_df_1[features + ['mass']], train_df_1['signal'])
```

```
Train a UGradientBoostingClassifier with new features and removing some feature
s
```

Out[89]:
```
UGradientBoostingClassifier(learning_rate=0.15,
                            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 0x1AEE92DC940,
                            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 [90]:
```
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='id
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 0.07795205982595821
True
```

In [91]:
```
#check_correlation = pandas.read_csv(folder + 'check_correlation.csv', index_col=
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 0.0017555368714286441
True
```

In [92]:
```
test_probs = model.predict_proba(test_df_1[features])[:,1]
result = pd.DataFrame({"id": test_df["id"], "prediction": test_probs})
result.to_csv("result4a.csv", index=False)
```

Both tests pass and performance of this model with test data on kaggle is also improved. Let us try using a different algorithm of RandomForestClassifier with hyperparameter tuning

In [134]:
```
# hyperparameter tuning
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [100,200,500,600,800]
# Number of features to consider at every split
max_features = ['auto']
# Maximum number of levels in tree
max_depth = [5,10,20,40,80]
# Minimum number of samples required to split a node
min_samples_split = [5, 10, 20, 40]
# Minimum number of samples required at each leaf node
min_samples_leaf = [2, 4, 10]
# Method of selecting samples for training each tree
bootstrap = [True]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(train_df_1[features], train_df_1['signal'])
```

Out[134]:
```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.
0,
                                                    n_estimators=100,
                                                    n_jobs...
                                                    verbose=0,
                                                    warm_start=False),
                   iid='deprecated', n_iter=50, n_jobs=-1,
                   param_distributions={'bootstrap': [True],
                                        'max_depth': [5, 10, 20, 40, 80],
                                        'max_features': ['auto'],
                                        'min_samples_leaf': [2, 4, 10],
```

```
                                        'min_samples_split': [5, 10, 20, 40],
                                        'n_estimators': [100, 200, 500, 600,
                                                         800]},
                           pre_dispatch='2*n_jobs', random_state=42, refit=True,
                           return_train_score=False, scoring=None, verbose=0)
```

In [135]:
```
rf_random.best_estimator_
```

Out[135]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=40, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [70]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=40, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
model.fit(train_df_1[features], train_df_1['signal'])
```

Out[70]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=40, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=2, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [71]:
```
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='id
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 0.04498071333281184
True
```

In [72]:
```
#check_correlation = pandas.read_csv(folder + 'check_correlation.csv', index_col=
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 0.0009992148828636693
True
```

In [73]:
```
test_probs = model.predict_proba(test_df_1[features])[:,1]
result = pd.DataFrame({"id": test_df["id"], "prediction": test_probs})
result.to_csv("result5.csv", index=False)
```

Both the tests pass with RandomForest model but performance is not as greate as UGradientBoosting

Let us try different variations of feature engineering

In [74]:
```
# trying variations in operations to create some new features based on kaggle dis
def new_feats(df):
    df2 = df.copy()
    df2['isolation_abc'] = df['isolationa'] * df['isolationb'] * df['isolationc']
    df2['isolation_def'] = df['isolationd'] * df['isolatione'] * df['isolationf']
    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 'FlightDu=istance' and LifeTime(from literature)
    df2['FD_LT']=df['FlightDistance']/df['LifeTime']
    #new feature using 'FlightDistance', 'po_p', 'p1_p', 'p2_p'(from literature)
    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_Chi2Dof
    df2['Chi2Dof_MAX'] = df.loc[:, ['p0_track_Chi2Dof', 'p1_track_Chi2Dof', 'p2_t
    # features from kaggle discussion forum
    df2['flight_dist_sig2'] = (df['FlightDistance']/df['FlightDistanceError'])**2
    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=1)
    df2['iso_bdt_min'] = df.loc[:, ['p0_IsoBDT', 'p1_IsoBDT', 'p2_IsoBDT']].min(a
    df2['iso_min'] = df.loc[:, ['isolationa', 'isolationb', 'isolationc','isolati
    return df2
```

In [75]:
```
train_df_1 = new_feats(train_df)
test_df_1 = new_feats(test_df)
```

In [76]:
```
remove = ['id', 'min_ANNmuon', 'production', 'mass', 'signal',
                'SPDhits','CDF1', 'CDF2', 'CDF3',
                'isolationb', 'isolationc','p0_pt', 'p1_pt', 'p2_pt',
                'p0_p', 'p1_p', 'p2_p', 'p0_eta', 'p1_eta', 'p2_eta',
                'isolationa', 'isolationb', 'isolationc', 'isolationd', 'isolatione
                'p0_IsoBDT', 'p1_IsoBDT', 'p2_IsoBDT',
                'p0_IP', 'p1_IP', 'p2_IP',
                'IP_p0p2', 'IP_p1p2',
                'p0_track_Chi2Dof', 'p1_track_Chi2Dof', 'p2_track_Chi2Dof',
                'p0_IPSig', 'p1_IPSig', 'p2_IPSig',
                'DOCAone', 'DOCAtwo', 'DOCAthree']
features = list(f for f in train_df.columns if f not in remove)
```

In [77]:
```
print("Train a UGradientBoostingClassifier with new features and removing some fe
loss = BinFlatnessLossFunction(['mass'], n_bins=15, uniform_label=0 , fl_coeffici
model = UGradientBoostingClassifier(loss=loss, n_estimators=900,
                                    max_depth=6,
                                    learning_rate=0.15,
                                    train_features=features,
                                    subsample=0.7)
model.fit(train_df_1[features + ['mass']], train_df_1['signal'])
```

```
Train a UGradientBoostingClassifier with new features and removing some feature
s
```

Out[77]:
```
UGradientBoostingClassifier(learning_rate=0.15,
                            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 0x1AEE92DC940,
                            splitter='best', subsample=0.7,
                            train_features=['LifeTime', 'dira',
                                            'FlightDistance',
                                            'FlightDistanceError', 'IP',
                                            'IPSig', 'VertexChi2', 'pt', 'iso',
                                            'ISO_SumBDT'],
                            update_tree=True)
```

In [78]:
```
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='id
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 0.04186936644380568
True
```

In [79]:
```
#check_correlation = pandas.read_csv(folder + 'check_correlation.csv', index_col=
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 0.0009566844499362014
True
```

In [80]:
```
test_probs = model.predict_proba(test_df_1[features])[:,1]
result = pd.DataFrame({"id": test_df["id"], "prediction": test_probs})
result.to_csv("result9.csv", index=False)
```

The model with slight changes in features passes the tests but does not perform as good as previous model. So sticking with the UGraientBoosting algorithm itself

In [93]:
```
def new_feats(df):
    df2 = df.copy()
    df2['isolation_abc'] = df['isolationa'] + df['isolationb'] + df['isolationc']
    df2['isolation_def'] = df['isolationd'] + df['isolatione'] + df['isolationf']
    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 'FlightDu=istance' and LifeTime(from literature)
    df2['FD_LT']=df['FlightDistance']/df['LifeTime']
    #new feature using 'FlightDistance', 'po_p', 'p1_p', 'p2_p'(from literature)
    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_Chi2Dof
    df2['Chi2Dof_MAX'] = df.loc[:, ['p0_track_Chi2Dof', 'p1_track_Chi2Dof', 'p2_t
    # features from kaggle discussion forum
    df2['flight_dist_sig2'] = (df['FlightDistance']/df['FlightDistanceError'])**2
    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=1)
    df2['iso_bdt_min'] = df.loc[:, ['p0_IsoBDT', 'p1_IsoBDT', 'p2_IsoBDT']].min(a
    df2['iso_min'] = df.loc[:, ['isolationa', 'isolationb', 'isolationc','isolati
    return df2


train_df_1 = new_feats(train_df)
test_df_1 = new_feats(test_df)

remove = ['id', 'min_ANNmuon', 'production', 'mass', 'signal','SPDhits','CDF1',
          'p0_pt', 'p1_pt', 'p2_pt','p0_p', 'p1_p', 'p2_p', 'p0_eta', 'p1_eta',
          'isolationc', 'isolationd', 'isolatione', 'isolationf','p0_IsoBDT', 'p1
          'p2_IP','IP_p0p2', 'IP_p1p2','p0_track_Chi2Dof', 'p1_track_Chi2Dof', 'p
          'p2_IPSig','DOCAone', 'DOCAtwo', 'DOCAthree']
features = list(f for f in train_df_1.columns if f not in remove)

#len(features)

#print("Train a UGradientBoostingClassifier with new features and removing some f
loss = BinFlatnessLossFunction(['mass'], n_bins=15, uniform_label=0 , fl_coeffici
model = UGradientBoostingClassifier(loss=loss, n_estimators=1000,
                                    max_depth=7,
                                    learning_rate=0.15,
                                    train_features=features,
                                    subsample=0.7)
model.fit(train_df_1[features + ['mass']], train_df_1['signal'])
```

Out[93]:
```
UGradientBoostingClassifier(learning_rate=0.15,
                            loss=BinFlatnessLossFunction(allow_wrong_signs=Tr
ue,
                                                         fl_coefficient=15,
                                                         n_bins=15, power=2,
                                                         uniform_features=['m
```

```
ass'],
                                                    uniform_label=array
([0])),
                            max_depth=7, max_features=None, max_leaf_nodes=No
ne,
                            min_samples_leaf=1, min_samples_split=2,
                            n_estimators=1000,
                            random_state=RandomState(MT19937) at 0x1AEE92DC94
0,
                            splitte...
                            train_features=['LifeTime', 'dira',
                                            'FlightDistance',
                                            'FlightDistanceError', 'IP',
                                            'IPSig', 'VertexChi2', 'pt', 'is
o',
                                            '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_sig
2',
                                            'flight_dist_sig', 'NEW_IP_dira',
                                            'p0p2_ip_ratio', 'p1p2_ip_ratio',
                                            'DCA_MAX', 'iso_bdt_min',
                                            'iso_min'],
                            update_tree=True)
```

In [94]:
```
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='id
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 0.07434361234769682
True
```

In [95]:
```
#check_correlation = pandas.read_csv(folder + 'check_correlation.csv', index_col=
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 0.0021117647116199053
False
```

In [ ]:
```
test_probs = model.predict_proba(test_df_1[features])[:,1]
result = pd.DataFrame({"id": test_df["id"], "prediction": test_probs})
result.to_csv("result9.csv", index=False)
```