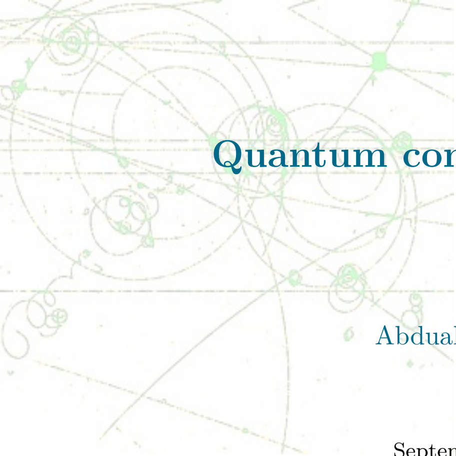


Quantum computing in HEP

Abdualazem Fadol

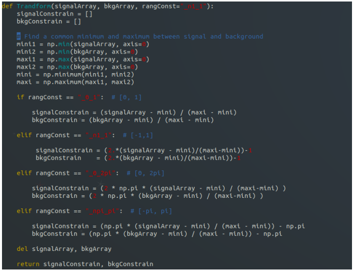
September 2, 2022

中国科学院高能物理研究所
Institute of High Energy Physics
Chinese Academy of Sciences

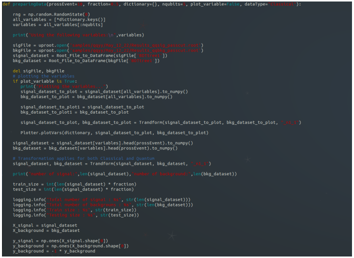
ATLAS
EXPERIMENT

A quick recap of the framework

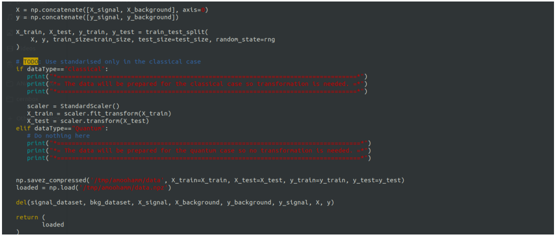
- □ Memory leakage problem was solved:
    - ○ fix the mess on transform
    - ○ restructure the data pre-processing
    - ○ sync the data to files
    - ○ using GridSearchCV (leads to another problem)
- □ Based on Qhep_Modules/Utilities.py
- □ So far Qiyu Sha tested this successfully



```python
def Trandform(signalArray, bkgArray, rangConst="_n1_1"):
    signalConstrain = []
    bkgConstrain = []

    # Find a common minimum and maximum between signal and background
    mini1 = np.min(signalArray, axis=0)
    mini2 = np.min(bkgArray, axis=0)
    maxi1 = np.max(signalArray, axis=0)
    maxi2 = np.max(bkgArray, axis=0)
    mini = np.minimum(mini1, mini2)
    maxi = np.maximum(maxi1, maxi2)

    if rangConst == "_0_1":  # [0, 1]

        signalConstrain = (signalArray - mini) / (maxi - mini)
        bkgConstrain = (bkgArray - mini) / (maxi - mini)

    elif rangConst == "_n1_1":  # [-1,1]

         signalConstrain = (2.*(signalArray - mini)/(maxi-mini))-1
         bkgConstrain    = (2.*(bkgArray - mini)/(maxi-mini))-1

    elif rangConst == "_0_2pi":  # [0, 2pi]

        signalConstrain = (2 * np.pi * (signalArray - mini) / (maxi-mini) )
        bkgConstrain = (2 * np.pi * (bkgArray - mini) / (maxi-mini) )

    elif rangConst == "_npi_pi":  # [-pi, pi]

        signalConstrain = (np.pi * (signalArray - mini) / (maxi - mini)) - np.pi
        bkgConstrain = (np.pi * (bkgArray - mini) / (maxi - mini)) - np.pi

    del signalArray, bkgArray

    return signalConstrain, bkgConstrain
```

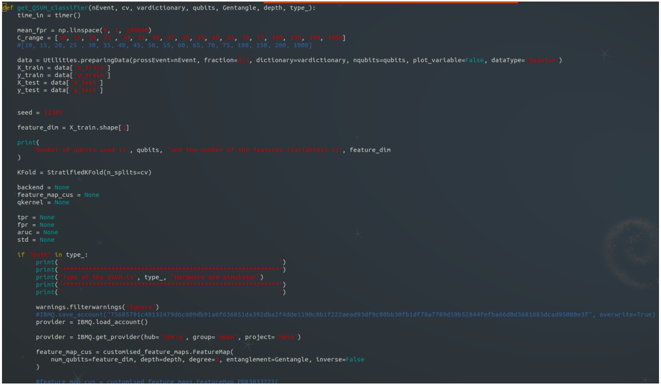


```python
def preparingData(prossEvent=10, fraction=0.5, dictionary={}, nqubits=5, plot_variable=False, dataType="Classical"):

    rng = np.random.RandomState(0)
    all_variables = [*dictionary.keys()]
    variables = all_variables[:nqubits]

    print("Using the following variables:\n",variables)

    sigFile = uproot.open('samples/qqyy/May_12_22/Results_qqsig_passcut.root')
    bkgFile = uproot.open('samples/qqyy/May_12_22/Results_qqbkg_passcut.root')
    signal_dataset = Root_File_to_DataFrame(sigFile['BDTtree1'])
    bkg_dataset = Root_File_to_DataFrame(bkgFile['BDTtree1'])

    del sigFile, bkgFile
    # plotting the variables
    if plot_variable is True:
       print("Plotting the variables...")
       signal_dataset_to_plot = signal_dataset[all_variables].to_numpy()
       bkg_dataset_to_plot = bkg_dataset[all_variables].to_numpy()

       signal_dataset_to_plot1 = signal_dataset_to_plot
       bkg_dataset_to_plot1 = bkg_dataset_to_plot

       signal_dataset_to_plot, bkg_dataset_to_plot = Trandform(signal_dataset_to_plot, bkg_dataset_to_plot, "_n1_1")

       Plotter.plotVars(dictionary, signal_dataset_to_plot, bkg_dataset_to_plot)

    signal_dataset = signal_dataset[variables].head(prossEvent).to_numpy()
    bkg_dataset = bkg_dataset[variables].head(prossEvent).to_numpy()

    # Transformation applies for both Classical and Quantum
    signal_dataset, bkg_dataset = Trandform(signal_dataset, bkg_dataset, "_n1_1")

    print("number of signal:",len(signal_dataset),"number of background:",len(bkg_dataset))

    train_size = int(len(signal_dataset) * fraction)
    test_size = int(len(signal_dataset) * fraction)

    logging.info("Total number of signal : %s", str(len(signal_dataset)))
    logging.info("Total number of backgrouns : %s", str(len(bkg_dataset)))
    logging.info("Train size : %s", str(train_size))
    logging.info("Testing size : %s", str(test_size))

    X_signal = signal_dataset
    X_background = bkg_dataset

    y_signal = np.ones(X_signal.shape[0])
    y_background = np.ones(X_background.shape[0])
    y_background = -1 * y_background
```



```python
    X = np.concatenate([X_signal, X_background], axis=0)
    y = np.concatenate([y_signal, y_background])

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, train_size=train_size, test_size=test_size, random_state=rng
    )

    # TODO: Use standarised only in the classical case
    if dataType=="Classical":
       print("*==================================================================================*")
       print("*= The data will be prepared for the classical case so transformation is needed. =*")
       print("*==================================================================================*")

       scaler = StandardScaler()
       X_train = scaler.fit_transform(X_train)
       X_test = scaler.transform(X_test)
    elif dataType=="Quantum":
       # Do nothing here
       print("*===================================================================================*")
       print("*= The data will be prepared for the quantum case so no transformation is needed. =*")
       print("*===================================================================================*")


    np.savez_compressed('/tmp/amoohamm/data', X_train=X_train, X_test=X_test, y_train=y_train, y_test=y_test)
    loaded = np.load('/tmp/amoohamm/data.npz')

    del(signal_dataset, bkg_dataset, X_signal, X_background, y_background, y_signal, X, y)

    return (
           loaded
    )
```

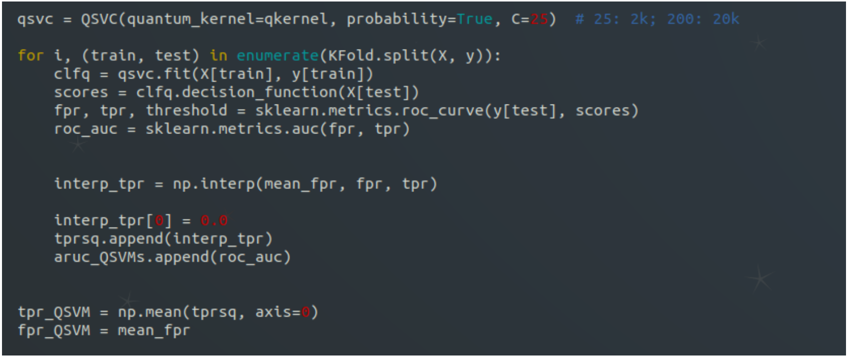
A quick recap of the framework



```python
def get_QSVM_classifier(nEvent, cv, vardictionary, qubits, Gentangle, depth, type_):
    time_in = timer()

    mean_fpr = np.linspace(0, 1, 100000)
    C_range = [10, 15, 20, 25 , 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 100, 150, 200, 1000]
    #[10, 15, 20, 25 , 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 100, 150, 200, 1000]

    data = Utilities.preparingData(prossEvent=nEvent, fraction=0.5, dictionary=vardictionary, nqubits=qubits, plot_variable=False, dataType="Quantum")
    X_train = data['X_train']
    y_train = data['y_train']
    X_test = data['X_test']
    y_test = data['y_test']


    seed = 12345

    feature_dim = X_train.shape[1]

    print(
        "Number of qubits used is", qubits, "and the number of the features (variables) is", feature_dim
    )

    KFold = StratifiedKFold(n_splits=cv)

    backend = None
    feature_map_cus = None
    qkernel = None

    tpr = None
    fpr = None
    aruc = None
    std = None

    if "both" in type_:
        print("                                                          ")
        print("**********************************************************")
        print("Type of the QSVM is", type_, "Hardware and simulator")
        print("**********************************************************")
        print("                                                          ")

        warnings.filterwarnings('ignore')
        #IBMQ.save_account("75685791c48132479d6c609db91a6f636051da392dba2f4dde1190c8b1f222aead93df9c00bb30fb1df78a7789d59b52844fefba66d0d3681683dcad95080e3f", overwrite=True)
        provider = IBMQ.load_account()

        provider = IBMQ.get_provider(hub='ibm-q', group='open', project='main')

        feature_map_cus = customised_feature_maps.FeatureMap(
            num_qubits=feature_dim, depth=depth, degree=1, entanglement=Gentangle, inverse=False
        )

        #feature_map_cus = customised_feature_maps.FeatureMap_PRR3033221(
```

- ☐ Adding two modules:
  - ○ Qhep_Modules/QSVM_class.py
  - ○ Qhep_Modules/SVM_class.py
- ☐ Combining everything in bin/classification.py
- ☐ Using single line:
  - ○ python bin/classification.py -e Parforward -n 100 -d 1 -t both -q 6
  - ○ python bin/classification.py -e Parforward -n 100 -d 1 -t simulator -q 6

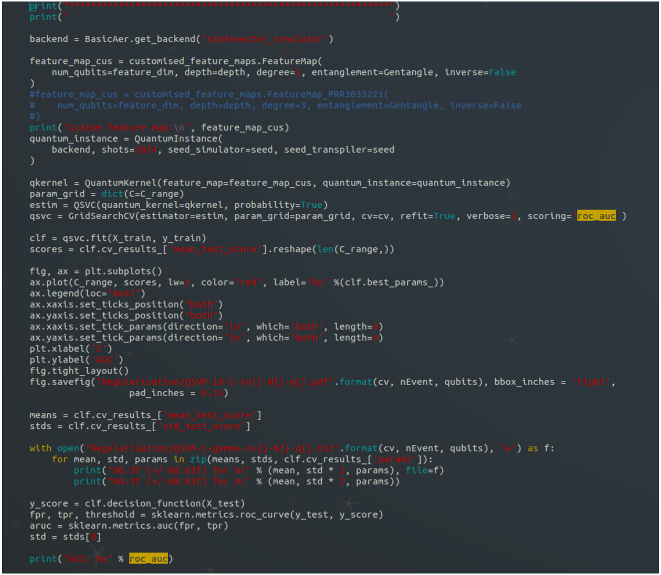
The GridSearchCV ROC curve problem



```
qsvc = QSVC(quantum_kernel=qkernel, probability=True, C=25)  # 25: 2k; 200: 20k

for i, (train, test) in enumerate(KFold.split(X, y)):
    clfq = qsvc.fit(X[train], y[train])
    scores = clfq.decision_function(X[test])
    fpr, tpr, threshold = sklearn.metrics.roc_curve(y[test], scores)
    roc_auc = sklearn.metrics.auc(fpr, tpr)


    interp_tpr = np.interp(mean_fpr, fpr, tpr)

    interp_tpr[0] = 0.0
    tprsq.append(interp_tpr)
    aruc_QSVMs.append(roc_auc)


tpr_QSVM = np.mean(tprsq, axis=0)
fpr_QSVM = mean_fpr
```

- □ The old way of doing the training (memory overload).
- □ The AUC from the scan is different from the one out of the fixed parameters.

The GridSearchCV ROC curve problem



```
print("***********************************************************")
print("                                                           ")

backend = BasicAer.get_backend("statevector_simulator")

feature_map_cus = customised_feature_maps.FeatureMap(
    num_qubits=feature_dim, depth=depth, degree=1, entanglement=Gentangle, inverse=False
)
#feature_map_cus = customised_feature_maps.FeatureMap_PRR3033221(
#    num_qubits=feature_dim, depth=depth, degree=3, entanglement=Gentangle, inverse=False
#)
print("Custom feature map:\n", feature_map_cus)
quantum_instance = QuantumInstance(
    backend, shots=1024, seed_simulator=seed, seed_transpiler=seed
)

qkernel = QuantumKernel(feature_map=feature_map_cus, quantum_instance=quantum_instance)
param_grid = dict(C=C_range)
estim = QSVC(quantum_kernel=qkernel, probability=True)
qsvc = GridSearchCV(estimator=estim, param_grid=param_grid, cv=cv, refit=True, verbose=1, scoring='roc_auc')

clf = qsvc.fit(X_train, y_train)
scores = clf.cv_results_['mean_test_score'].reshape(len(C_range,))

fig, ax = plt.subplots()
ax.plot(C_range, scores, lw=1, color="red", label="%s" %(clf.best_params_))
ax.legend(loc="best")
ax.xaxis.set_ticks_position("both")
ax.yaxis.set_ticks_position("both")
ax.xaxis.set_tick_params(direction='in', which='both', length=6)
ax.yaxis.set_tick_params(direction='in', which='both', length=6)
plt.xlabel('C')
plt.ylabel('AUC')
fig.tight_layout()
fig.savefig("Regularisation/QSVM-1d-C-cv{}-N{}-q{}.pdf".format(cv, nEvent, qubits), bbox_inches = 'tight',
                  pad_inches = 0.15)

means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']

with open("Regularisation/QSVM-c-gamma-cv{}-N{}-q{}.txt".format(cv, nEvent, qubits), "w") as f:
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params), file=f)
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))

y_score = clf.decision_function(X_test)
fpr, tpr, threshold = sklearn.metrics.roc_curve(y_test, y_score)
aruc = sklearn.metrics.auc(fpr, tpr)
std = stds[0]

print("AUC: %s" % roc_auc)
```

Additional slides