Przygotowanie narzędzi do uczenia maszynowego opartego na danych z eksperymentu CMS.

Paweł Czajka Mateusz Fila Rafał Masełek

Opiekun projektu: dr hab. A. Kalinowski

Cel projektu

Celem projektu było przygotowanie narzędzi do efektywnego wykorzystania metod uczenia maszynowego (ang. *Machine Learning* – ML) do analiz fizycznych w eksperymencie Compact Muon Solenoid (CMS) przy Wielkim Zderzaczu Hadronów (LHC) w ośrodku badawczym CERN pod Genewą. Bazowym założeniem było wykorzystanie isniejącej analizy dedykowanej rozpadom bozonu Higgsa na parę taonów ($H \rightarrow \tau\tau$), wyekstraktowanie z niej interesujących obserwabli do plików w formacie czytelnym z poziomu języka programowania Python, a następnie wykorzystanie zaawansowanych bibliotek dedykowanych do uczenia maszynowego w celu poprawienia wydajności analizy.

1 Upublicznienie kodu projektu

Dla projektu stworzono dedykowane repozytorium na serwisie GitHub. Znajduje się ono pod adresem: `https://github.com/Rav2/HEPMachineLearning`. Początkowa część projektu polegała na modyfikacji analizy do rozpadów $H \rightarrow \tau\tau$, dostępnej pod adresem `https://github.com/akalinow/RootAnalysis`.

2 Modyfikacja istniejącej analizy

Pierwszym krokiem w budowie zintegrowanego narzędzia do analiz fizycznych z użyciem ML była modyfikacja istniejącej analizy dra Kalinowskiego.

2.1 Instalacja

Pliki C++ stworzone w ramach projektu, zostały zintegrowane z analizą dra Kalinowskiego i są dostępne pod adresem `https://github.com/akalinow/RootAnalysis`. Instrukcja kompilacji i użycia znajduje się w pliku README.md w katalogu głównym repozytorium. Po skompilowaniu analizy nie jest potrzebna żadna dodatkowa instalacja modułu ML. Warto nadmienić, iż analiza wymaga oprogramowania takiego jak: Pythia8[1], ROOT[2], Python[3] 2.7.x oraz biblioteki Boost[4] dla języka C++.

2.2 Założenia

Przyjęto następujące wytyczne:

- Kod powinien być zgodny ze standardem C++11.
- Modyfikacja powinna mieć formę modułu, nie zakłócać ani nie spowalniać normalnego działania analizy. Powinna istnieć możliwość wyłączenia działania modułu.
- Kod powinien być na tyle elastyczny, aby mógł być potencjalnie użyty do różnych analiz. Powinna istnieć możliwość wyboru parametrów zapisywanych do pliku.
- Sterowanie parametrami programu powinno odbywać się bez potrzeby ponownej kompilacji kodu, powinno dać się w razie potrzeby zautomatyzować. Zdecydowano się na użycie zewnętrznego pliku tekstowego z ustawieniami.
- Kod powinien być udokumentowany i napisany zgodnie z tzw. zasadami dobrego programowania.

- Należy stworzyć skrypt czytający powstałe pliki rootowe do języka Python.

3 Przepływ danych

1. Działanie programu ropoczyna się od wczytania pliku konfiguracyjnego, np. *HTauTau/Analysis/config/htt_MuTau.ini*. Znajdują się tam podstawowe parametry programu, takie jak: liczba wątków, folder na wyniki, ścieżka do plików wejściowych z danymi z eksperymentu CMS oraz ich nazwy.

2. Funkcja main analizy $H \rightarrow \tau\tau$ znajduje się w pliku *HTauTau/Analysis/src/HTTAnalysis.cc*. Wykonywane są w niej kolejno:

   (a) Wczytanie pliku konfiugracyjnego (np. *htt_MuTau.ini*) oraz parsowanie parametrów.

   (b) Utworzenie użytecznych obiektów:

   - **std::vector<Analyzer*> myAnalyzers** – kontener na obiekty dziedziczące po klasie *Analyzer*, tj. obiekty analizy implementujące selekcję danych i obliczenia. W dalszej części dokumentu będziemy nazywać takie obiekty po prostu *analizami*.
   - **EventProxyHTT *myEvent** – obiekt pośredniczący pomiędzy analizami a danymi.
   - **ObjectMessenger* OMess** – obiekt służący do przekazywania wyników obliczeń pomiędzy różnymi analizami.

   (c) Obiekty analiz są tworzone i umieszczane w kontenerze *myAnalyzers*. Jako ostatnia jest dodawana analiza przygotowująca dane do obróbki algorytmami ML[1]. Klasa za to odpowiedzialna nazywa się *MLAnalyzer* i dziedziczy po klasie *Analyzer*. Wymaga do pracy obiektu klasy *MLObjectMessenger* dziedziczącego po klasie *ObjectMessenger*.

   (d) Następnie tworzony jest obiekt **TreeAnalyzer *tree**, który będzie sterował wykonywaniem analiz. Obiekt ma dostęp do konte-

[1] Głównym zadaniem analizy ML jest przygotowanie i zapisanie danych do pliku .root. Ponieważ ten typ plików jest wykorzystywany jedynie w trybie pracy jednowątkowej, dlatego analiza ML nie zostanie utworzona, jeśli program zostanie uruchomiony w trybie wielowątkowym.

nera z analizami, obiektu typu *ObjectMessenger* oraz pliku konfiguracyjnego.

(e) W końcowym etapie działania funkcji następuje wypisanie statystyk na stdout oraz zwolnienie pamięci.

3. W obiekcie klasy TreeAnalyzer tworzone są histogramy wspólne dla wszystkich analiz. W metodzie *int TreeAnalyzer::loop()* znajduje się pętla iterująca po wszystkich *eventach* (elementarnych zderzeniach). Dla każdego eventu wykonywane są analizy z kontenera *myAnalyzers* w kolejności w jakiej zostały tam dodane. Wykorzystywana jest do tego metoda *bool TreeAnalyzer::analyze(const EventProxyBase&)*.

4. ObjectMessenger jest klasą zaopatrzoną w kontenery do przechywania danych w wybranych typach podstawowych. Dane są umieszczane i pobierane z kontenerów za pomocą odpowiednich metod. Ze względu na swoją elastyczność, funkcjonalność klasy ObjectMessenger może zostać poszerzona o nowe typy i dedykowane kontenery. Jedną z takich klas jest klasa MLObjectMessenger, która gromadzi wyniki pracy analiz aby dostarczyć je do analizy ML.

5. Analiza ML wykonuje następujące czynności:

   (a) W trakcie tworzenia obiektu wczytywany jest plik *HTauTau/Analysis/config/ml_Properties.ini*, w którym użytkownik może wybrać, które wielkości powinny znaleźć się w pliku wyjściowym.

   (b) Dane pobrane z obiektu MLObjectMessenger są przygotowywane do zapisu. W pliku wyjściowym tworzona jest wewnętrzna struktura.

   (c) Czteropędy zrekonstruowanych cząstek i jetów są zapisywane do pliku wyjściowego.

   (d) Pozostałe parametry są zapisywane.

4 Lista plików C++

Poniżej znajduje się lista plików C++ utworzonych bądź znacząco zmodyfikowanych w trakcie realizacji pierwszej części projektu:

- **MLAnalyzer.h** – plik nagłówkowy dla klasy MLAnalyzer dziedziczącej po klasie Analyzer. Zadaniem klasy jest wczytanie ustawień z pliku

ml_Properties.ini, utworzenie struktury drzewa w pliku .root, zapisanie odpowiednich danych do tego pliku.

- **MLAnalyzer.cc** – plik źródłowy dla klasy MLAnalyzer.
- **ObjectMessenger.h** – plik nagłówkowy dla klasy *ObjectMessenger*. Obiekty tej klasy są kontenerami, w których przechowywane są dane przesyłane pomiędzy różnymi analizami. Klasa posiada interfejsy do modyfikacji i odczytu danych dla najczęściej używanych typów podstawowych.
- **ObjectMessenger.cc** – plik źródłowy dla klasy *ObjectMessenger*.
- **MLObjectMessenger.h** – plik nagłówkowy dla klasy *MLObjectMessenger* dziedziczącej po klasie *ObjectMessenger*. W stosunku do klasy bazowej jest wzbogacona o obsługę typów *HTTParticle* oraz *HTTAnalysis::sysEffects*.
- **ml_Properties.ini** – plik tekstowy umożliwiający wybór predefiniowanych parametrów, które zostaną zapisane do drzewa w pliku .root. Dane z pliku są wczytywane w metodzie *MLAnalyzer::ParseCfg(const std::string&)*

4.1 Konwersja danych z pliku ROOT do Pythona

Po wyekstraktowaniu danych z analizy do pliku .root należy wczytać je do części frameworku napisanej w języku programowania Python. W dalszej części opisany zostanie moduł user_defined_function.py, w której można zaimplementować funkcję get_data(), z której korzysta już reszta pythonowego frameworku. Najwygodniej chyba osiągnąć to korzystając z funkcji które eksportują dane z formatu .root do plików w formacie pickle. TODO napisac gdzie one sa.

5 Uczenie maszynowe w Python

5.1 Założenia pythonowych programów

- Chcieliśmy osiągnąć ogólność naszego kodu. Powinien on działać zarówno dla problemów binarnej klasyfikacji jak i regresji. W obecnej wersji można użyć dowolnych numerycznych featcherów oraz featcherów kategorycznych.

- Podejście funkcyjne. Jedynymi efektami ubocznymi napisanych funkcji są powstawanie plików oraz zmiany wag modelu. Dzięki temu kod można łatwiej zrozumieć oraz modyfikować.

- Wszystkie ważne użyte funkcje mają dokumentację umieszczoną w komentarzach.

- Staraliśmy się zastosować do wytycznych dotyczących wydajności kodu ze strony https://www.tensorflow.org/guide/performance/datasets . Dzięki temu trenowanie zachodzi szybko.

- W programie użyte są standardowe estymatory tensorflowowe używające głębokich sieci neuronowych tf.estimator.DNNClassifier oraz tf.estimator.DNNClass Dzięki temu łatwo można zmodyfikować kod tak, by korzystać z innych estymatorów tensorflowowych (być może własnoręcznie napisanych) gdyż wszystkie estymatory mają podobne funkcje train, evaluate oraz predict.

5.2 Pliki pythonowe

Powstały następujące pliki pythonowe

- **io_TFRecords** Ten moduł umożliwia tworzenie plików binarnych w tensorflowowym formacie TFRecord, który jest wspomniany w dokumentacji tensorflow. Ten binarny format danych pozwala modelom tensorflowowym uczyć się na dużych zbiorach danych w szybki sposób. Obsługa tego formatu jest jednakże dość skomplikowana, gdyż trzeba pamiętać dokładnie jakiego typu dane były zapisane w pliku binarnym by można było go odczytać. Z tego powodu moduł ten tworzy foldery w których oprócz binarnych danych w formacie TFRecord znajdują się pliki z informacją o rodzajach zapisanych danych, które umożliwiają wygodne odczytanie danych. W plikach można zapisywać numeryczne featchery które są listami floatów (ustalonej długości) oraz featchery kategoryczne w postaci liczb całkowitych (wówczas dostarczamy klasie także możliwe wartości naszych kategorycznych danych). Przewidywana wielkość może być zarówno binarną informacją typu tło-sygnał jak również może być przewidywaną liczbą liczbą rzeczywista. Dodatkowo wraz z danymi używanymi do trenowania można zapisać dane które będą przez estymatory ignorowane. Dzięki temu można później sprawdzać korelacje tych nie używanych featcherów z uzyskanymi wynikami.

- **model_dnn** Ten moduł zawiera obudowane standardowe estymatory tensorflowowe w taki sposób, by mogły one łatwo uczyć się na datasetach zapisanych przy pomocy modułu io_TFRecords. W zależności od tego, czy problem polega na binarnej klasyfikacji czy regresji użyty jest estymator tf.estimator.DNNClassifier lub tf.estimator.DNNRegressor. Ta klasa zajmuje się także normalizacją danych wchodzących do estymatora.

- **create_tf_records_folder** Ten moduł korzysta z modułu io_TFRecords by zapisać dane w postaci formatu TFRecords. Korzysta on także z pliku user_defined_function.py, w którym znajduje się jedyna zależna od problemu funkcja, którą musi zaimplementować użytkownik. Funkcja ta zwraca dane które mają zostać zapisane.

5.3 Użycie pythonowych klas

5.3.1 user_defined_function.py

W tym pliku należy zaimplementować funkcję get_data(). Funkcja ta ma za zadanie przekazać dane featcherów numerycznych oraz kategorycznych. W tym miejscu użytkownik decyduje się które featchery mają zostać użyte do uczenia, a które mają występować tylko w celu analizy wyników. Tutaj również należy określić, czy powstający dataset ma służyć do binarnej klasyfikacji czy do regresji. Dokładniejsza dokumentacja tej funkcji znajduje się w komentarzu na początku pliku create_tf_records_folder.py.

5.3.2 create_tf_records_folder.py

W tym pliku zaimplementowana jest funkcja create_tfr_record która tworzy przy pomocy modułu io_TFRecords plik z danymi zwróconymi prez funkcję user_defined_function.get_data(). Moduł ten zajmuje się wstępnym tasowaniem tych danych oraz automatycznie rozponaje jakie są zbiory możliwych wartosci danych które użytkownik wskazał jako kategoryczne.

5.3.3 model_dnn.py

Użytkownik chcący wytrenować model będzie korzystał z następujących funkcji (mają one dokładną dokumentację w postaci komentarzy w kodzie)

- create_model_folder, która pozwala utworzyć model tensorflowowy oraz wybrać jego architekturę.

- train pozwalająca trenować model na folderze danych utworzonym przez klasę io_TFRecords
- evaluate pozwalająca ewaluować działanie modelu. Ta funkcja używa również z danych zapisanych przy pomocy klasy io_TFRecords
- load_model jest istotna, jeśli chcemy samodzielnie zaimplementować jakąś funkcjonalność. Wczytuje ona tensorflowowy model zapisany w określonym folderze.
- input_fn jest również istotna dla osób modyfikujących działanie tego modułu. Wszystkie estymatory muszą przyjmować dane w postaci odpowiedniej funkcji zwanej input function. Można tą funkcję zmodyfikować do swoich potrzeb.

5.3.4 io_TFRecords

Z tego modułu użytkownik nie musi bezpośredni korzystać, jednakże warto przytoczyć funcjonalność w celu ułatwienia ewentualnej modyfikacji.

- create_empty_data_folder tworzy pusty folder danych. Należy jej powiedzieć jakiego typu danych się spodziewamy w tym folderze (to znaczy podajemy rozmiary featcherów oraz możliwe wartości featcherów kategorycznych)
- write_one_example zapisuje pojedynczy przykład do folderu
- parse_individualy_not_ignored pozwala odczytac tensorflow.dataset zawierający dane które będą używane podczas uczenia. Z tej funkcji korzysta model_dnn.
- parse_batch_all pozwala odczytać zarówno dane ignorowane (czyli nie używane podczas uczenia) jak i nie ignorowane. Jest ona użyteczna jeśli ktos chce porównać wartość tych ignroowanych wielkości z predykcjami modelu.

Literatura

[1] Torbjorn Sjostrand, Stephen Mrenna, and Peter Z. Skands. A Brief Introduction to PYTHIA 8.1. *Comput. Phys. Commun.*, 178:852–867, 2008.

[2] R. Brun and F. Rademakers. ROOT: An object oriented data analysis framework. *Nucl. Instrum. Meth.*, A389:81–86, 1997.

[3] Python org. webpage. `https://www.python.org`. Accessed: 2019-01-21.

[4] Boost library webpage. `https://www.boost.org`. Accessed: 2019-01-21.