Uniwersytet Warszawski
Wydział Fizyki

Agnieszka Ciepielewska
Nr albumu: 385537

Zastosowanie uczenia maszynowego do identyfikacji leptonów tau w eksperymencie CMS

Praca licencjacka
na kierunku Fizyka w ramach Międzywydziałowych Indywidualnych Studiów Matematyczno-Przyrodniczych

Praca wykonana pod kierunkiem
dr hab. Artur Kalinowski
Zakład Cząstek i Oddziaływań Fundamentalnych
Instytut Fizyki Doświadczalnej

Warszawa, lipiec 2019

Oświadczenie kierującego pracą

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

Leptony tau odgrywają istotną rolę w wielu eksperymentach fizycznych. Niestety ze względu na krótki czas życia obserwuje się zazwyczaj produkty ich rozpadu, które mogą być mylone z dżetami hadronowymi. W poniższej pracy przedstawiono modele uczenia maszynowego zastosowane do identyfikacji taonów. Najlepszym modelem okazał się *XGBoost*, który był lepszy niż wytrenowane sieci neuronowe oraz wcześniej stosowane klasyfikatory.

Słowa kluczowe

leptony τ, CMS, uczenie maszynowe, sieci neuronowe, XGBoost

Dziedzina pracy (kody wg programu Socrates-Erasmus)

13.2 Fizyka

Tytuł pracy w języku angielskim

Application of machine learning for tau leptons identification in the CMS experiment

Spis treści

Cel pracy . . . 3

1. Wstęp teoretyczny . . . 4
1.1. Leptony tau . . . 4
1.2. Detektor CMS . . . 5
1.3. Sieci neuronowe . . . 6
1.3.1. Funkcje aktywacji . . . 7
1.3.2. Warstwa *batch normalization* . . . 8
1.3.3. Harmonogram stałej uczenia . . . 8
1.4. XGBoost . . . 8
1.5. Miary skuteczności modeli . . . 9
1.6. Istotność zmiennych . . . 10

2. Dane symulacyjne . . . 11

3. Architektura modeli . . . 15
3.1. Model oparty o same klasyfikatory . . . 15
3.2. Sieć neuronowa . . . 15
3.3. Poprawiona sieć neuronowa . . . 15
3.4. XGBoost . . . 17

4. Wyniki . . . 18

5. Dyskusja . . . 24

6. Podsumowanie . . . 25

Bibliografia . . . 25

Cel pracy

Leptony τ to cząstki o krótkim czasie życia, także trudno jest wykrywać je bezpośrednio. Z tego powodu zazwyczaj obserwuje się produkty ich rozpadu. Ze względu na dużą masę taonów, mogą one rozpadać się hadronowo, co czyni je trudnymi do odróżnienia od dżetów hadronowych. W Wielkim Zderzaczu Hadronów w eksperymencie CMS zbierane są dane ze zderzeń proton-proton, w których mogą być produkowane cząstki rozpadające się na taony takie jak bozon Higgsa. Danych jest na tyle dużo, że nie mogą być przeglądane przez człowieka w celu znalezienia obserwacji w których wystąpił rozpad taonu. Dlatego w tym celu stosowane jest uczenie maszynowe, które pozwala na wstępną selekcję danych.

Jednakże dane rzeczywiste nie dostarczają informacji czy w zderzeniu wystąpił taon, także nie da się zastosować uczenia nadzorowanego na takich danych. Dlatego stosuje się dane symulacyjne, gdzie metodami Monte Carlo symulowane są rozpady $H \to \tau\tau$ jako sygnał oraz dżety hadronowe jako tło.

Celem pracy było stworzenie modeli uczenia maszynowego pozwalających na rozróżnienie leptonów τ od dżetów hadronowych oraz porównanie wytrenowanych modeli z już istniejącymi klasyfikatorami.

Rozdział 1

Wstęp teoretyczny

Poniżej podano podstawowe informacje dotyczące taonów, detektora CMS oraz uczenia maszynowego, które są później używane.

1.1. Leptony tau

Leptony tau odgrywają istotną rolę w wielu eksperymentach fizycznych. Jednym z ważniejszych jest badanie bozonu Higgsa w rozpadzie $H \to \tau\tau$. Jednak z powodu krótkiego czasu życia $\tau = 2.9 \cdot 10^{-13}$ s [2], one również nie są wykrywane bezpośrednio w eksperymentach, ale obserwuje się ich produkty rozpadu. Taony są najcięższymi z leptonów i przy swojej masie $m_\tau = 1.776$ GeV/c^2 [2] jako jedyne są w stanie rozpadać się hadronowo. Z tabeli 1.1 widać, że dzieje się tak w około 2/3 przypadków. Sprawia to problemy przy ich identyfikacji, ponieważ łatwo jest je pomylić z dżetami hadronowymi (ang. *QCD jets*).

Tabela 1.1: Główne kanały rozpadu taonów wraz z prawdopodobieństwem [6, 2]. Tutaj h oznacza zarówno mezony π jak i K.

Rozpad	Prawdopodobieństwo [%]
Rozpady leptonowe	
$\tau^- \to e^- \bar{\nu}_e \nu_\tau$	17.8
$\tau^- \to \mu^- \bar{\nu}_\mu \nu_\tau$	17.4
Rozpady hadronowe	
$\tau^- \to h^- \pi^0 \nu_\tau$	25.9
$\tau^- \to h^- \nu_\tau$	11.5
$\tau^- \to h^- h^+ h^- \nu_\tau$	9.8
$\tau^- \to h^- \pi^0 \pi^0 \nu_\tau$	9.5
$\tau^- \to h^- h^+ h^- \pi^0 \nu_\tau$	4.8
Inne	3.3

1.2. Detektor CMS

Detektor CMS (*Compact Muon Solenoid*) znajduje się w Wielkim Zderzaczu Hadronów (LHC) [1] w CERN-ie. Ilustracja obrazująca wygląd CMS znajduje się na rysunku 1.1. Jego główne elementy to: nadprzewodząca cewka wytwarzająca pole magnetyczne o indukcji 3.8 T, detektory krzemowe, kalorymetr elektromagnetyczny (ECAL), kalorymetr hadronowy (HCAL) i komora jonizacyjna do wykrywania mionów.

Rysunek 1.1: Schemat budowy detektora CMS.

Detektor krzemowy, pokrywający przedział pseudopospieszności[1] $|\eta| < 2.5$, składa się z dwóch rodzajów detektorów: pikselowych i paskowych. Korpus centralny składa się z trzech warstw detektorów pikselowych oraz jedenastu warstw detektorów paskowych, natomiast końcówki z jedenastu dysków, z czego dwa zawierają detektory pikselowe, a pozostałe detektory paskowe [3]. Tor lotu hadronów jest rekonstruowany ze skutecznością 80-90% w zależności od pędu poprzecznego i pseudopospieszności η. Detektory krzemowe mają grubość od 0.4 do 2.0 długości drogi radiacyjnej (X_0), także fotony z dużym prawdopodobieństwem konwertują wewnątrz detektora na pary e^+e^- [6].

Kalorymetr ECAL jest zrobiony ze scyntylacyjnych kryształów stolzytu ($PbWO_4$). Posiada ponad 61000 kryształów w korpusie centralnym, pokrywającym $|\eta| < 1.48$ oraz ponad 7300 kryształów na końcach detektora, pokrywających $|\eta| < 3.0$. Stolzyt posiada krótką drogę radiacyjną ($X_0 = 0.89$ cm) oraz szybko wyświeca światło (80% światła jest wyświecane w 25 ns). Dzięki temu dobrze rozdziela kaskady, więc jest często wykorzystywany do budowy

[1] Pseudopospieszność η cząstki definiowana jest jako $\eta = -\ln \mathrm{tg}(\theta/2)$, gdzie θ to kąt między kierunkiem lotu cząstki a kierunkiem wiązki.

kalorymetrów [3].

Na zewnątrz ECAL znajduje się kalorymetr hadronowy HCAL zawierający mosiądz i plastik. Mosiądz został wybrany ze względu na krótką drogę swobodną (ang. *interaction length*) hadronów oraz słabe właściwości magnetyczne. Tak samo jak ECAL pokrywa obszar $|\eta| < 3.0$. Jego grubość to od siedmiu do jedenastu dróg swobodnych w zależności od η. Natomiast dla $3.0 < \eta < 5.0$ używany jest stalowo kwarcowy kalorymetr przedni (ang. *Hadron Forward*) [3].

Detektor mionowy składa się z trzech rodzajów komór gazowych. W części centralnej ($|\eta| < 1.2$) używane są komory *drift tube* (DT), w części końcowej ($|\eta| < 2.4$), pole magnetyczne nadal jest duże, ale niejednolite, używa się *cathode strip chambers* (CSC). Dodatkowo we wszystkich miejscach są zastosowane *resistive plate chambers* (RPC). RPC zapewnia dobrą dokładność czasową, natomiast DT i CSC dobrą dokładność pozycyjną [3].

1.3. Sieci neuronowe

Sieci neuronowe to klasa modeli uczenia maszynowego stosowana w uczeniu nadzorowanym [9]. Na podstawie danych treningowych zawierający zmienne objaśniające i objaśniane model uczy się zależności występujących między tymi zmiennymi. Dzięki temu możliwe jest stosowanie modelu do predykcji zmiennych objaśnianych na podstawie nowych obserwacji.

Sieci neuronowe składają się z warstw. Podstawową warstwą używaną w sieciach neuronowych jest warstwa gęsta (ang. *dense layer* lub *fully connected layer*). Warstwa gęsta przyjmuje na wejściu wektor, a na wyjściu zwraca nowy wektor ustalonej wielkości, niekoniecznie tego samego rozmiaru. Jest to warstwa bardzo generyczna, jednak jej minusem jest duża liczba parametrów. Każda warstwa gęsta składa się z dwóch podstawowych elementów: transformacji liniowej oraz funkcji nieliniowej, zwanej funkcją aktywacji. Każda transformacja liniowa posiada wagi w_i oraz przesunięcia b_i (ang. *bias*). Wyjście z warstwy definiowane jest jako:

$$y = f(x^T W + b),$$

gdzie x to wektor wejściowy, f - funkcja aktywacji, nie zmieniająca wymiaru wektora, W - macierz wag warstwy, b - wektor przesunięcia. Dodanie kolejnych warstw to złożenie kolejnych takich funkcji.

Przykładowo na rysunku 1.2 widoczne jest wejście do sieci x, które jest wektorem 3-elementowym. Następnie jest on transponowany i przemnażany przez macierz wag W_1, która jest wymiaru 3×4 oraz dodane jest przesunięcie b_1 o wymiarze 1×4. Na tym etapie nakłada się również funkcję aktywacji f_1. Kółka w warstwie ukrytej reprezentują wyjścia transformacji liniowych już po zastosowaniu funkcji aktywacji. Kolejny krok jest analogiczny, tym razem jest to jedna regresja liniowa, czyli macierz wag W_2 ma wymiar 4×1, a przesunięcie b_2 1×1. Zatem wyjście y:

$$y = f_2\left(\left(f_1(x^T W_1 + b_1)\right)^T W_2 + b_2\right),$$

gdzie f_1, f_2 to funkcje aktywacji. Zauważmy, że nieliniowe funkcje aktywacji są konieczne, gdyż inaczej wielowarstwową sieć neuronową zawsze dałoby się sprowadzić do przypadku zwykłej transformacji liniowej odpowiednio przemnażając wagi [9].

Rysunek 1.2: Przykładowa sieć neuronowa z jedną ukrytą warstwą gęstą.

Następnie aby otrzymać optymalny model poszukiwane jest minimum pewnej funkcji straty L (czyli miejsce, gdzie model najmniej się myli). Przez funkcję straty rozumiana jest funkcja przybliżająca błąd popełniany przez model. Minimum jest znajdowane przy użyciu gradientu funkcji L względem parametrów modelu. Niestety nie da się wyznaczyć gradientu L w każdym miejscu, ale można go znaleźć w punkcie, gdzie dokonywana jest predykcja. Także, aby poprawić wagi oraz przesunięcie w modelu używa się iteracyjnego algorytmu *Gradient Descent*, który polega na obliczeniu gradientu funkcji straty po parametrach modelu $\nabla_{w_i} L$, a następnie aktualizowane są wagi o pewien krok w kierunku ujemnego gradientu, także dla przykładowej wagi w przy $n+1$-szej iteracji (górny indeks oznacza numer iteracji):

$$w^{(n+1)} \leftarrow w^{(n)} - \gamma \frac{\partial L^{(n)}}{\partial w^{(n)}},$$

gdzie γ to tzw. stała uczenia (ang. *learning rate*) [9]. Widać, że istotne jest aby wszystkie funkcje użyte w modelu były różniczkowalne prawie wszędzie (w punktach nieciągłości pochodnej można przyjąć dowolną wartość brzegową).

W praktyce sieci neuronowe uczone są algorytmem *Stochastic Gradient Descent* (SGD). Polega on na liczeniu funkcji straty dla pewnej liczby obserwacji (zwanej później *batchem*), zamiast dla wszystkich obserwacji. Jednorazowe przejście przez wszystkie obserwacje nazywane jest epoką, a zazwyczaj trening modelu składa się z wielu epok. Przykładowy wygląd krzywej uczenia, czyli wykresu funkcji straty od liczby obserwacji przedstawiony jest na rysunku 1.3.

Rozwinięciem algorytmu SGD jest algorytm *Adam* (*Adaptive momentum*) [11], który pomaga w zapobieganiu pozostawania w lokalnych minimach funkcji straty.

1.3.1. Funkcje aktywacji

Stosowane później funkcje aktywacji wraz ze wzorami:

- Sigmoid:

$$S(x) = \frac{1}{1+e^{-x}},$$

Rysunek 1.3: Przykładowa zmiana funkcji straty dla zbioru treningowego i testowego od liczby obserwacji, pionowymi liniami zaznaczono końce kolejnych epok.

- ReLU[12]

$$ReLU(x) = \max(0, x).$$

1.3.2. Warstwa *batch normalization*

Wraz z postępem w treningu sieci wyjścia z warstw sieci mogą się zmieniać. W szczególności rozkład wyjścia może ulec zmianie, do czego każda następna warstwa sieci musi się na nowo dostosowywać. Przeciwdziała temu warstwa *batch normalization*, która przeskalowuje każdy batch tak aby miał tą samą średnią i odchylenie standardowe [10]. Dzięki zastosowaniu *batch normalization* trening sieci trwa krócej, a sieć może osiągnąć lepszą skuteczność.

1.3.3. Harmonogram stałej uczenia

Poza dopasowywaniem wielkości kroku aktualizacji wag zastosowanym w algorytmie *Adam*, stosuje się harmonogram stałej uczenia. Polega on na zmniejszaniu stałej uczenia, gdy występuje wypłaszczenie krzywej uczenia. Jeśli funkcja straty nie zmniejsza się przez określoną liczbę epok, stała uczenia zmniejszana jest o pewien czynnik aż osiągnie zadaną wcześniej wartość minimalną.

1.4. XGBoost

W uczeniu maszynowym przez drzewo decyzyjne rozumie się model statystyczny dokonujący predykcji na podstawie sekwencji binarnych decyzji (rysunek 1.4). W praktyce pojedyncze drzewo decyzyjne jest rzadko stosowane ze względu na niską skuteczność predykcyjną. W zamian używane są algorytmy, które budują wiele drzew decyzyjnych i agregują ich wyniki.

Jednym z nich jest XGBoost (*eXtreme Gradient Boosting*) [5], który jest implementacją algorytmu *Gradient Boosting*. W pierwszym kroku budowane jest pojedyncze drzewo decyzyjne, które możliwie najlepiej przewiduje zmienną objaśnianą. Następnie, każde kolejne drzewo jest tworzone aby jak najlepiej przewidywać błąd popełniany przez poprzedni zestaw drzew. Końcowa predykcja to suma wszystkich odpowiedzi drzew.

Rysunek 1.4: Przykładowe drzewo decyzyjne, a, b, c to zmienne występujące w danych

1.5. Miary skuteczności modeli

Jako, że identyfikacja taonów to problem klasyfikacji binarnej, dobrą metryką do mierzenia skuteczności modelu jest ROC AUC (*Area Under Receiver Operating Characteristic Curve*) oraz sama krzywa ROC. Analizowany model przewiduje prawdopodobieństwo, czy dany rozpad zawiera taon. Aby dokonać klasyfikacji musimy ustalić próg odcięcia ponad którym zawsze zwracane jest 1 (wystąpił taon), a poniżej 0 (nie wystąpił). Wówczas można policzyć prawdziwe pozytywa (ang. *true positives*, TP), czyli obserwacje oznaczone w danych jako sygnał i przewidziane jako sygnał oraz fałszywe pozytywa (ang. *false positives*, FP), czyli obserwacje oznaczone w danych jako tło, a przewidziane jako sygnał. Następnie można obliczyć *true positive rate* (TPR), czyli TP podzielone przez liczbę wszystkich obserwacji oznaczonych w danych jako sygnał oraz *false positive rate* (FPR), czyli FP podzielone przez liczbę wszystkich obserwacji oznaczonych w danych jako tło. Krzywa ROC to wykres TPR od FPR dla różnych progów odcięcie. Jednak zgodnie z występującą w fizyce cząstek konwencją, na wykresach krzywej ROC osie są zamienione, zatem jest to wykres FPR od TPR.

ROC AUC to pole pod krzywą ROC. W związku z tym przyjmuje ona wartości pomiędzy 0 a 1, gdzie 0.5 to wartość osiągana przez model losowy (przewidującego losowe prawdopodobieństwo np. z rozkładu jednostajnego od 0 do 1), zaś 1 przez bezbłędny klasyfikator. Przykładowa krzywa ROC oraz jej ROC AUC dla modelu losowego oraz nielosowego znajdują się na rysunku 1.5. ROC AUC nie jest różniczkowalna, więc nie można jej bezpośrednio wykorzystać przy treningu sieci neuronowej jako funkcji straty. Jest jednak jest przydatna przy ewaluacji oraz porównywaniu wytrenowanych modeli.

Do treningu jako funkcje straty wykorzystuje się zazwyczaj binarną entropię krzyżową

(ang. *binary cross entropy*) [9], daną wzorem:

$$L(y, p) = -(y \log(p) + (1 - y) \log(1 - p)),$$

gdzie y to prawdziwe prawdopodobieństwo, że dana obserwacja zawiera taon (0 albo 1), a p to jego oszacowanie przez model.

Rysunek 1.5: Przykładowe krzywe ROC dla klasyfikatora losowego oraz nielosowego. W nawiasach kwadratowych podano wartość ROC AUC dla danego modelu.

1.6. Istotność zmiennych

Jednym ze sposobów sprawdzenia istotności zmiennych w modelu predykcyjnym jest metoda permutacyjna [4]. Polega na losowym permutowaniu jednej zmiennej objaśniającej w zbiorze danych, a następnie mierzeniu jaką skuteczność osiąga model na takim zbiorze. Jeśli skuteczność modelu się pogorszyła, wnioskuje się, że dana zmienna jest używana przez model przy predykcji. Po wykonaniu tej operacji dla każdej zminnej objaśniającej można uzyskać informacje które zmienne są w największym stopniu wykorzystywane przez model.

Rozdział 2

Dane symulacyjne

Dane użyte w pracy pochodzą z symulacji zderzeń proton-proton w których został wyprodukowany bozon Higgsa i rozpadł się na parę $\tau\tau$ (szukany sygnał). Natomiast za tło zostały przyjęte dżety QCD.

Przy analizowaniu rozpadów taonów wydziela się części przestrzeni, które są potencjalnie bardziej interesujące. Takimi częściami są stożek sygnału (ang. *signal cone*) oraz stożek izolacji (ang. *isolation cone*). W stożku sygnału powinny znajdować się jedynie produkty rozpadu taonu. Na rysunku 2.1 znajduje się przykładowy rozpad taonu z zaznaczonym stożkiem sygnałowym oraz izolacji.

Rysunek 2.1: Przykładowy schemat rozpadu taonu z zaznaczonym stożkiem sygnałowym oraz izolacji.

Dane są wstępnie przetwarzane tak, aby pozostawić tylko najbardziej potrzebne informacje. Ostatecznie, otrzymane dane mają 19 zmiennych objaśniających [7]:

- **byCombinedIsolationDeltaBetaCorrRaw3Hits** - Suma pędów cząstek naładowanych oraz fotonów wewnątrz stożka izolacji pomniejszona o korektę $\Delta\beta$, zapisuje się wzorem:
$$I_\tau = \sum p_T^{\text{charged}}(d_z < 0.2\text{cm}) + \max(0, \sum p_T^\gamma - \Delta\beta)$$
.
- **chargedIsoPtSum** - Suma pędów naładowanych produktów rozpadu wewnątrz stożka sygnałowego.
- **decayDistMag** - Odległość między zderzeniem pp a punktem, w którym nastąpił rozpad taonu.
- **decayMode** - Kanał rozpadu taonu.
- **dxy** - Odległość między zderzeniem pp a punktem, w którym nastąpił rozpad taonu w rzucie na płaszczyznę prostopadłą do osi wiązki.
- **dxy_Sig** - Zmienna **dxy** podzielona przez niepewność pomiaru.
- **eRatio** - Stosunek energii elektromagnetycznej do całkowitej energii całkowitej produktów rozpadu.
- **flightLengthSig** - Zmienna **decayDistMag** podzielona przez niepewność pomiaru.
- **gjAngleDiff** - Kąt między kierunkiem ruchu taonu oraz kierunkiem ruchu wiodącego produktu rozpadu.
- **hasSecondaryVertex** - Informacja czy rozpad posiada wierzchołek wtórny, którym może być rozpad taonu.
- **ip3d** - Odległość między zderzeniem pp a najbliższym punktem na krzywej zakreślanej przez wiodący produkt rozpadu taonu.
- **nPhoton** - Liczba protonów o pędzie $p_T > 0.5$ GeV przypisana do rozpadu taonu.
- **neutralIsoPtSum** - Suma pędów neutralnych produktów rozpadu wewnątrz stożka sygnałowego.
- **photonPtSumOutsideSignalCone** - Suma pędów fotonów poza stożkiem sygnałowym.
- **ptWeightedDetaStrip** - Suma $\Delta\eta$ ważona pędami cząstek, gdzie $\Delta\eta$ to różnica η między taonem a produktem rozpadu taonu.
- **ptWeightedDphiStrip** - Suma $\Delta\phi$ ważona pędami cząstek, gdzie $\Delta\phi$ to różnica ϕ między taonem a produktem rozpadu taonu.
- **ptWeightedDrIsolation** - Suma ΔR ważona pędami cząstek wewnątrz stożka izolacji, gdzie $\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}$.
- **ptWeightedDrSignal** - Suma ΔR ważona pędami cząstek wewnątrz stożka sygnałowego.

- **puCorrPtSum** - Suma pędów cząstek neutralnych znajdujących się wewnątrz stożka sygnałowego, ale pochodząca z innych rozpadów.

Dodatkowo dane zawierają 4 zmienne będące odpowiedziami wcześniej wykorzystywanych klasyfikatorów do identyfikacji taonów:

- **byIsolationMVArun2v1DBnewDMwLTraw2017v2** - Model drzewiasty oparty o algorytm *AdaBoost* [8].
- **deepTau2017v1tauVSall** - Jeden z modeli opartych o głębokie sieci neuronowe. Używany do rozróżniania taonów od wszystkich rodzajów tła.
- **deepTau2017v1tauVSjet** - Jeden z modeli opartych o głębokie sieci neuronowe. Używany do rozróżniania taonów od dżetów QCD.
- **DPFTau_2016_v1tauVSall** - Jeden z modeli opartych o głębokie sieci neuronowe. W przeciwieństwie do pozostałych modeli, był trenowany na danych zawierających pojedyncze produkty rozpadu (a nie zsumowane).

Jako, że są to dane symulacyjne, występuje tam również informacja o wystąpieniu taonu, czyli zmienna objaśniania. Rozpatrujemy tylko jeden rodzaj tła, także przyjmuje ona wartość 1 jeśli taon wystąpił (sygnał) oraz 0 jeśli był to dżet QCD (tło).

Na rysunku 2.2 przedstawiono rozkłady przykładowych zmiennych dla sygnału oraz tła. Widać zauważalne różnice w rozkładach, także można na podstawie tej zmiennej dokonywać klasyfikacji. Rozkłady częściowo się nakładają, więc klasyfikacja na podstawie żadnej ze zmiennych nie byłaby idealna.

(a) decayMode

(b) eRatio

(c) ptWeightedDrIsolation

(d) ptWeightedDetaStrip

(e) gjAngleDiff

(f) nPhoton

Rysunek 2.2: Rozkłady przykładowych zmiennych dla sygnału i tła

Rozdział 3

Architektura modeli

Aby dokonywać klasyfikacji sygnału zastosowano różne modele predykcyjne opierające się na metodach opisanych w rozdziale 1. Modele zostały napisane w języku *python* z wykorzystaniem bibliotek *keras* oraz *xgboost*. Kod jest dostępny w repozytorium: `https://github.com/byrolew/licencjat`.

3.1. Model oparty o same klasyfikatory

Prosty model, składający się z jednego neuronu, który przyjmuje jako zmienne objaśniające odpowiedzi wcześniej używanych klasyfikatorów (patrz rozdział 2). Jest to więc średnia ważona odpowiedzi klasyfikatorów, na którą nałożona jest funkcja sigmoid. Wielkość batcha została ustalona na 128, do optymalizacji został użyty algorytm *Adam* ze stałą uczenia $\gamma = 10^{-3}$. Model uczony był przez cztery epoki. Później używana jest nazwa *ensemble*.

3.2. Sieć neuronowa

Kolejny model był oparty o wszystkie zmienne objaśniające, jego architekturę przedstawiono na rysunku 3.1. Składał się z sześciu warstw gęstych, o wielkości 32 neuronów każda. Jako funkcje aktywacji zastosowano funkcje ReLU. Warstwę wyjściową tworzy jeden neuron z sigmoidem jako funkcją aktywacji. Dzięki temu wyjściem sieci jest pojedyncza wartość z zakresu [0, 1]. Inne parametry zostały ustalone tak samo jak dla modelu w sekcji 3.1. Ten model zwany jest później jako *baseline*.

3.3. Poprawiona sieć neuronowa

Na podstawie optymalizacji hiperparametrów przy pomocy losowego przeszukiwania został wybrany najlepszy model oparty o gęste sieci neuronowe. Optymalizowano parametry takie jak: wielkość batcha, stała uczenia, wielkość i liczba warstw ukrytych, funkcja aktywacji, liczba epok, występowanie warstwy *batch normalization* i występowanie harmonogramu stałej uczenia. Otrzymana architektura znajduje się na rysunku 3.2.

Wybrany model ma dwie warstwy gęste po 256 neuronów każda, zatem widać że model jest płytszy i szerszy niż opisany w sekcji 3.2. Jako funkcja aktywacji jest tutaj użyty sigmoid, a dodatkowo zastosowana jest warstwa *batch normalization*. Użyta wielkość batcha to 256, do optymalizacji zastosowano algorytm *Adam* ze stałą uczenia $\gamma = 5 \cdot 10^{-4}$ oraz harmonogram zmniejszający stałą uczenia γ na wypłaszczeniu funkcji straty przez czynnik 0.2 do minimalnej

wartości $\gamma = 10^{-5}$. Model był uczony przez sześć epok. Ten model zwany jest później jako *best-nn*.

Rysunek 3.1: Bazowa sieć neuronowa (*baseline*). Składa się z 6 ukrytych warstw gęstych po 32 neurony. Jako funkcji aktywacji użyto ReLU.

Rysunek 3.2: Zoptymalizowana sieć neuronowa (*best-nn*). Składa się z dwóch ukrytych warstw gęstych oraz warstwy *batch normalization*. Jako funkcji aktywacji użyto sigmoida.

3.4. XGBoost

Model oparty na XGBoost składał się ze 100 drzew decyzyjnych o maksymalnej głębokości 3. Wykorzystano stałą uczenia $\gamma = 10^{-1}$, funkcją kosztu była binarna entropia krzyżowa (tak samo jak w przypadku sieci neuronowych). Schematyczny rysunek architektury znajduje się na rysunku 3.3.

Rysunek 3.3: Architektura modelu typu XGBoost opartego na drzewach decyzyjnych. Tutaj użyto modelu zawierającego 100 drzew o maksymalnej głębokości 3 (nie licząc korzenia).

Rozdział 4

Wyniki

Każdy model trenowany był w dwóch trybach. Pierwszy z nich korzystał ze wszystkich zmiennych objaśniających, drugi zaś pomijał odpowiedzi klasyfikatorów. Dzięki temu można porównać wyniki z wcześniej używanymi modelami. Po dokonaniu analizy istotności zmiennych metodą permutacyjną zauważono, że zmienna byCombinedIsolationDeltaBetaCorrRaw3Hits (patrz rozdział 2) jest najbardziej istotna dla wszystkich modeli. Jako, że rozkład tej zmiennej różni się na danych symulacyjnych oraz eksperymentalnych, postanowiono wytrenować również modele bez tej zmiennej.

W tabeli 4.1 znajdują się wyniki ROC AUC dla wszystkich modeli. Dla porównania zamieszczono wynik najlepszego z wcześniejszych klasyfikatorów, którym okazał się deepTau2017v1tauVSjet (później nazywany *deepTau*).

Można zauważyć, że model oparty o XGBoost osiąga najwyższą skuteczność spośród zaproponowanych oraz wcześniej używanych modeli. W szczególności przy treningu na danych z wyłączeniem klasyfikatorów zarówno *best-nn* jak i *XGBoost* osiągnęły lepsze wyniki niż *deepTau*. Widać również, że wyrzucenie głównej zmiennej nie pogarsza znacząco skuteczności modeli, a może skutkować lepszymi wynikami na danych eksperymentalnych. Ponadto, model oparty o same klasyfikatory osiąga lepszy wynik niż każdy klasyfikator osobno.

Tabela 4.1: Wartość ROC AUC dla wszystkich modeli na zbiorze testowym, pogrubioną czcionką zaznaczono najlepszy wynik.

Tryb	*baseline*	*best-nn*	*XGBoost*	*deepTau*	*ensemble*
Pełne dane	.9948	.9979	**.9985**	–	–
Bez klas.	.9940	.9949	**.9956**	.9945	–
Bez najlepszej	.9977	.9972	**.9985**	–	–
Same klas.	–	–	–	–	.9956

Na rysunku 4.1 przestawiono rozkłady odpowiedzi modeli dla sygnału oraz tła. Widać, że rozkłady wyglądają podobnie dla wszystkich modeli, maksimum dla sygnału przypada dla odpowiedzi modelu równej 1, a dla tła dla odpowiedzi równej 0. Można zauważyć, że histogram gęstości odpowiedzi modelu osiąga mniejsze wartości dla odpowiedzi nieprawidłowych na obserwacjach oznaczonych w danych jako tło niż na tych oznaczonych w danych jako sygnał. Oznacza to, że model jest bardziej *pewny* swoich odpowiedzi dla tła niż dla sygnału. Dodatkowo dla modeli *ensemble* oraz *baseline* w rozkładzie odpowiedzi dla sygnału widoczna jest górka w okolicy 0. Oznacza to, że dla części obserwacji oznaczonych w danych jako sygnał, model jest pewny, że jest to tło, co skutkuje niewykryciem taonu.

(a) *ensemble*

(b) *baseline*

(c) *best-nn*

(d) *XGBoost*

(e) *deepTau*

Rysunek 4.1: Rozkłady odpowiedzi modeli dla sygnału i tła. Modele *ensemble*, *baseline* oraz *XGBoost* uczone były na pełnych danych.

Krzywe uczenia dla poszczególnych modeli znajdują się na rysunku 4.2. Dla modeli opartych o sieci neuronowe wykresy przestawiają spadek funkcji straty w zależności od liczby obserwacji na jakiej były uczone. Dodatkowo pionowymi liniami zaznaczono końce kolejnych epok. Natomiast dla *XGBoosta* wykres pokazuje spadek funkcji straty wraz ze wzrostem liczby drzew.

Dla wszystkich modeli widać różnice między wynikiem na zbiorze treningowym, na którym model jest uczony oraz testowym, który służy jedynie do pomiaru skuteczności modeli. Są one niewielkie, co oznacza, że model dobrze generalizuje na nieznane przykłady. Różnice między tymi modelami najlepiej widać na rysunku 4.3, gdzie przedstawiono wartość funkcji straty na zbiorze testowym dla modeli opartych na sieciach. Model *XGBoost* nie jest na nim przedstawiony, ponieważ jego krzywa uczenia nie może być bezpośrednio porównywana z krzywymi uczenia sieci neuronowych.

Można zauważyć, że funkcja straty dla modelu *ensemble* maleje wolniej niż dla modeli *baseline* oraz *best-nn*. Natomiast dla modelu *baseline* ma ona większą wariancję niż dla pozostałych. Krzywe mocno się wypłaszczają wraz z treningiem, jednak widać, że dla modeli *baseline* oraz *best-nn* potencjalnie trening można wydłużyć.

Na rysunkach 4.4, 4.5 i 4.6 przedstawiono fragmenty krzywych ROC dla wybranych modeli. Zakres TPR na wykresach został zmniejszony, aby lepiej uwidocznić różnice między modelami.

Rysunek 4.4 przedstawia krzywe ROC dla najlepszego modelu każdego typu. Można zauważyć, że największa różnica jest między *deepTau* a innymi modelami, natomiast najskuteczniejszy jest *XGBoost*. Pomimo tego, że różnice w ROC AUC są niewielkie dla wszystkich modeli, to jeśli punkt odcięcia zostałby wybrany tak, aby TPR był równy 0.5 (model wykrywa połowę taonów), to różnice FPR dla *XGBoosta* i *deepTau* to blisko jeden rząd wielkości.

Na rysunku 4.5 znajdują się wszystkie tryby treningu nowej sieci w porównaniu z *deepTau*. Wyniki dla treningu na pełnych danych i z wyłączeniem głównej zmiennej nie różnią się znacząco, natomiast trening bez klasyfikatorów widocznie pogarsza wygląd krzywej ROC. Na tym fragmencie osiąga gorsze wyniki niż *deepTau*. Jednak patrząc na wartość ROC AUC można podejrzewać, że osiąga lepsze rezultaty na skrajnych fragmentach.

Natomiast rysunek 4.6 pokazuje porównanie każdego typu modelu uczonego na danych bez klasyfikatorów. Dzięki temu można lepiej porównać skuteczność modeli, gdyż każdy z nich uczony był na takich samych danych. Można zauważyć, że na wyróżnionym fragmencie najlepsze wyniki osiąga *deepTau*, jednak są one zbliżone do wyników *XGBoosta*. Patrząc na wartości ROC AUC możemy przypuszczać, że zarówno *XGBoost* jak i *best-nn* uzyskują lepsze rezultaty niż *deepTau* na skrajnych zakresach TPR.

(a) *ensemble*

(b) *baseline*

(c) *best-nn*

(d) *XGBoost*

Rysunek 4.2: Zmiana wartości funkcji straty liczona na zbiorze testowym dla wszystkich modeli. Dla modeli opartych o sieci neuronowe, na osi x znajduje się liczba obserwacji, a dla *XGBoosta* liczba dopasowanych drzew. Modele *ensemble*, *baseline* oraz *XGBoost* uczone były na pełnych danych.

Rysunek 4.3: Zmiana wartości funkcji straty od liczby obserwacji liczona na zbiorze testowym dla wszystkich modeli opartych o sieci neuronowe.

Rysunek 4.4: Porównanie najlepszych modeli na podstawie krzywej ROC. *baseline*, *best-nn* oraz *XGBoost* były uczone na pełnych danych, a *deepTau* na danych bez klasyfikatorów. W nawiasach kwadratowych podano wartość ROC AUC dla danego modelu.

Rysunek 4.5: Porównanie wszystkich trybów uczenia nowej sieci z najlepszym klasyfikatorem na podstawie krzywej ROC. W nawiasach kwadratowych podano wartość ROC AUC dla danego modelu.

Rysunek 4.6: Porównanie modeli uczonych na danych bez klasyfikatorów na podstawie krzywej ROC. W nawiasach kwadratowych podano wartość ROC AUC dla danego modelu.

Rozdział 5

Dyskusja

Na podstawie przeprowadzonych eksperymentów najlepszym modelem okazał się *XGBoost*, którego wartość ROC AUC jest lepsza niż wytrenowanych sieci neuronowych oraz wcześniej stosowanych klasyfikatorów. Pomimo małych różnic ROC AUC wszystkie modele wytrenowane na pełnych danych okazały się skuteczniejsze niż wcześniej stosowane klasyfikatory. Najlepiej różnice widać na rysunku 4.4, gdzie dla założonego TPR, wartości FPR dla *XGBoosta* i *deepTau* różnią się mniej więcej o rząd wielkości.

Można jednak zauważyć, że krzywa uczenia modeli *baseline* oraz *best-nn* (rysunek 4.2) nie jest całkowicie wypłaszczona. Oznacza to, że trening tych modeli można by wydłużyć o kolejnych kilka epok i potencjalnie poprawić skuteczność modeli.

Warto też wspomnieć, że porównując dane symulacyjne i rzeczywiste widać różnice w rozkładach zmiennych objaśniających, także nie ma pewności, że model dający dobre wyniki na danych symulacyjnych będzie dawał równie dobre wyniki na danych rzeczywistych. Jednakże, sprawdzenie czy modele generalizują się na dane rzeczywiste wykracza poza ramy tej pracy.

Dodatkowo, klasyfikacja odbywała się tylko dla jednego rodzaju tła, czyli dżetów QCD. Należałoby rozszerzyć dane symulacyjne o różne rodzaje tła, aby sytuacja była bardziej zbliżona do danych rzeczywistych.

Rozdział 6

Podsumowanie

W pracy przedstawiono problem detekcji leptonów τ przy pomocy modeli uczenia maszynowego. Wprowadzono dwie klasy modeli, sieci neuronowe oraz modele oparte o drzewa decyzyjne. Sprawdzono ich skuteczność oraz porównano ze wcześniej stosowanymi klasyfikatorami. Okazało się, że model oparty na drzewach decyzyjnych (*XGBoost*) osiągnął najlepsze rezultaty. W szczególności wszystkie modele uczone na pełnych danych okazały się lepsze niż wcześniej używane klasyfikatory.

Bibliography

[1] Roman Adolphi et al. "The CMS experiment at the CERN LHC". In: *Jinst* 803 (2008), S08004.

[2] K Nakamura et al. "Review of Particle Physics". In: *Journal of Physics G: Nuclear and Particle Physics* 37.7A (2010), p. 075021. DOI: 10.1088/0954-3899/37/7a/075021. URL: https://doi.org/10.1088%2F0954-3899%2F37%2F7a%2F075021.

[3] GL Bayatian et al. *CMS Physics: Technical Design Report Volume 1: Detector Performance and Software*. Tech. rep. CMS-TDR-008-1, 2006.

[4] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[5] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM. 2016, pp. 785–794.

[6] CMS Collaboration. "Performance of reconstruction and identification of τ leptons decaying to hadrons and ν_τ in pp collisions at $\sqrt{s} = 13$ TeV". In: (2018). arXiv: 1809.02816 [hep-ex].

[7] CMS collaboration et al. "Reconstruction and identification of τ lepton decays to hadrons and $\nu\tau$ at CMS". In: *Journal of Instrumentation* 11.01 (2016), P01019.

[8] Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[10] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: (2015). arXiv: 1502.03167.

[11] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980.

[12] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: In *Proceedings of ICML'10*. 2010, pp. 807–814.