Uniwersytet Zielonogórski

Wydział Informatyki, Elektrotechniki i Automatyki

Praca dyplomowa

Kierunek: Informatyka

Analiza porównawcza bibliotek uczenia maszynowego języka C++ na potrzeby zastosowań w biostatystyce

inż. Kacper Wojciechowski

Promotor:
Prof. dr hab. inż. Dariusz Uciński

Pracę akceptuję:

.....
(data i podpis promotora)

Zielona Góra, czerwiec 2023

Streszczenie

Niniejsza praca ma na celu analizę i porównanie dostępnych w języku C++ bibliotek uczenia maszynowego, pod kątem ich zastosowania w pracy na danych biostatystycznych. W kolejnych rozdziałach czytelnik zapoznawany jest z:

- Ogólną postacią problemów napotykanych w procesie implementacji rozwiązań uczenia maszynowego;
- Charakterystyką wybranego zestawu danych biostatystycznych wykorzystanych do testów omawianych bibliotek;
- Typami oraz uzyskiwanymi wynikami wybranych pod kątem danych eksperymentalnych metod uczenia maszynowego w środowisku prototypowym;
- Bibliotekami Tensorflow, Shark, Caffe i PyTorch wraz z metodami implementacji poszczególnych metod wzorcowych;
- Zbiorczym podsumowaniem funkcjonalności oferowanych przez wyżej wymienione biblioteki.

Słowa kluczowe: uczenie maszynowe, C++, biblioteka, sieci neuronowe, głębokie uczenie maszynowe, płytkie uczenie maszynowe.

Spis treści

Spis rysunków

Spis tabel

Rozdział 1

Wstęp

1.1. Wprowadzenie

We współczesnym stanie techniki coraz częściej można spotkać się z urządzeniami i programami o inteligentnych funkcjach, takich jak predykcja zjawisk na podstawie zestawu danych, rozpoznawanie obrazu, analiza mowy, czy przetwarzanie języka naturalnego. Znajdują one zastosowanie w różnych dziedzinach codziennego życia, m.in. w medycynie. W zależności od potrzeb, techniki uczenia maszynowego można wykorzystać do zastosowań medycznych, jak np. rozpoznawanie komórek rakowych na skanach rezonansem magnetycznym, podejmowanie decyzji na podstawie zbioru objawów obecnych u pacjenta, lub przewidywanie norm związków naturalnie występujących w organiźmie ludzkim w zależności od okoliczności i wyników pomiarów.

Jedną z istotnych dziedzin medycyny jest biostatystyka, polegająca na wykorzystaniu analizy statystycznej do wnioskowania na podstawie zbiorów danych, takich jak rezultaty przeprowadzonych badań (np. morfologicznych, poziomu poszczególnych hormonów we krwii, itp.), informacji o nawykach żywieniowych oraz stylu życia pacjenta. Szczególnie istotną formą systemów operujących w tej dziedzinie są systemy eksperckie, wykorzystujące techniki płytkiego i głębokiego uczenia maszynowego w celu wspierania diagnozy stawianej przez wykwalifikowanych lekarzy.

U podstaw wyżej wymienionych zagadnień leży implementacja rozwiązań opartych o teorię uczenia maszynowego, oraz wszelkie związane z tym problemy. W związku z tym na przestrzeni lat powstało wiele gotowych narzędzi, takich jak biblioteki i *frameworki*, mające na celu wsparcie programistów w szybkim i prawidłowym wprowadzaniu rozwiązań sztucznej inteligencji na różne platformy docelowe oraz w różnych językach, począwszy od języka C++, przez Python, po środowiska takie jak Matlab.

Istotnym krokiem w przygotowywaniu oprogramowania wykorzystującego sztuczną inteligencję jest prawidłowy wybór wspomnianych wcześniej narzędzi dokonywany na etapie projektowania, tak, aby oferowały one możliwości adekwatne do wymagań funkcjonalnych. Niniejsza praca dokonuje analizy porównawczej bibliotek uczenia maszynowego dla języka C++ w kontekście zastosowań w dziedzinie biostatystyki, celem umożliwienia czytelnikowi trafnego wyboru odpowiedniego narzędzia do realizacji projektu badawczego.

1.2. Cel i zakres pracy

Celem pracy jest przeprowadzenie analizy i przygotwanie zestawienia bibliotek do uczenia maszynowego dla języka C++, obrazując przykłady bazujące na zestawie danych biostatystycznych.

Zakres pracy obejmował:

- Przegląd dostępnych bibliotek języka C++;
- Inżynierię i kształtowanie danych;
- Płytkie i głębokie uczenie nadzorowane;
- Kwestie wydajnościowe w dopasowywaniu i wdrażaniu modeli;
- Badania praktyczne w oparciu o zestaw danych medycznych i biologicznych.

1.3. Struktura pracy

Pierwszy rozdział przedstawia ogólnym zagadnieniem dotykanym przez pracę, począwszy dziedziny problemu i jej zastosowań, do istoty tematu pracy. Dodatkowo omawiany jest cel i zakres realizacji pracy, oraz jej strukturą.

Kolejny rozdział wprowadza czytelnika do tematu uczenia maszynowego, oraz napotykanych w nim problemów dotyczących złożoności obliczeniowej oraz zużycia zasobów. Stanowią one podstawę do zaproponowania języka C++ jako technologi wspierającej ich rozwiązanie przy pomocy bibliotek.

Tematem rozdziału trzeciego jest przygotowanie elementów testowych do wykorzystania w późniejszej analizie porównawczej. Składa się na nie wybranie i przygotowanie do zestawu danych biostatystycznych do procesu uczenia oraz wybrane wzorcowych rozwiązań. Czytelnik przeprowadzony jest przez normalizację danych i selekcję najlepiej dopasowanych regresorów, oraz zostaje zapoznany z przykładowymi wynikami rozwiązań wzorcowych.

Dalsza część pracy składa się z bloku omówienia i analizy wybranych bibliotek uczenia maszynowego pod kątem określonych kryteriów. Pierwszą z nich, opisaną w rozdziale czwartym, jest biblioteka TensorFlow autorstwa Google. W poszczególnych sekcjach poruszone zostały tematy takie jak możliwe sposoby wdrożenia utworzonych w niej modeli, wymagana forma danych, dostępne funkcjonalności związane z danymi oraz modelami, a także dostępność źródeł informacji na temat sposobu pracy z biblioteką.

Rozdział 2

Uczenie maszynowe w ujęciu praktycznym

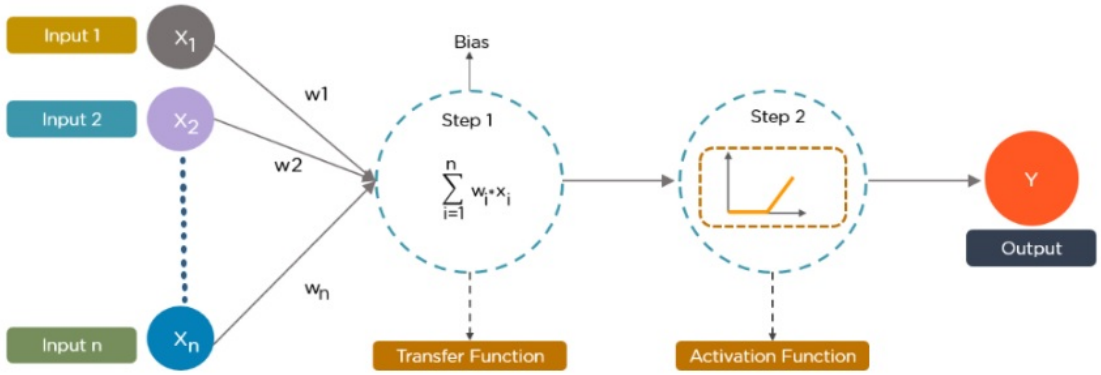
2.1. Problemy współczesnego uczenia maszynowego

Na uczenie maszynowe składają się zaawansowane techniki algorytmiczne i złożone struktury danych przeprowadzające obliczenia na zadanym przez użytkownika zestawie danych uczących, testujących, i danych otrzymywanych w trakcie użytkowania wytworzonego modelu.

Do podstawowych form modeli należą modele produkowane w wyniku technik takich jak regresja liniowa i nieliniowa, regresja logistyczna czy liniowa analiza dyskryminacyjna. W ich wyniku tworzone są modele w postaci wielomianów, które później wymagają stosunkowo bardzo małych nakładów mocy obliczeniowej w celu ewaluacji wyników na podstawie zadanego zestawu danych.

Bardziej zaawansowanymi metodami uczenia maszynowego są drzewa decyzyjne, stanowiące strukturę opartą o logikę drzewa. Każdy z poziomów drzewa odpowiada najlepszemu na danym etapie predyktorowi z dostępnych regresorów, powodując rozgałęzienie na poszczególne wartości lub zakresy. Proces obliczania wartości zmiennej wyjściowej odbywa się poprzez przejście przez drzewo od korzenia do jednego z końcowych liści.

Do najbardziej zaawansowanych, aczkolwiek także najbardziej wymagających obliczeniowo i pamięciowo technik uczenia maszynowego należą techniki uczenia głębokiego wykorzystujące sieci neuronowe, jak np. głębokie sieci neuronowe (ang. *Deep Neural Network, DNN*) i konwolucyjne sieci neuronowe (ang. *Convolutional Neural Network, CNN*). U podstaw tych metod leży struktura sieci neuronowej, składająca się z warstwy wejściowej, jednej lub więcej warstw ukrytych posiadających perceptrony, oraz jednej warstwy wyjściowej. Każdy węzeł z poprzedniej warstwy połączony jest z każdym węzłem w następnej warstwie, lecz perceptrony znajdujące się w tej samej warstwie są wzajemnie niezależne. Każde połączenie posiada przypisaną wagę użytą do przeliczenia wartości wchodzącej do danego perceptronu z danego sąsiada z poprzedniej warstwy. Wewnątrz peceptronu obliczana jest suma iloczynów wyjść z poprzednich perceptronów i wag odpowiadających połączeniom, a następnie dla uzyskanej sumy obliczana jest wartość funkcji aktywacyjnej, która stanowi wartość wyjściową perceptronu. Przykładowa sieć wykorzystująca pojedynczy perceptron w pojedynczej warstwie ukrytej przedstawiona została na rys. 2.1.



Rysunek 2.1. Schemat perceptronu - Simplelearn

Bardziej rozbudowane metody wykorzystujące sieci neuronowe, jak np. CNN, wymagają dodatkowych kroków obliczeniowych związanych z wstępnym przetworzeniem danych wejściowych, aby były one przyswajalne dla wykorzystywanej sieci.

Analizując struktury danych wymagane przez poszczególne omówione powyżej rodzaje modeli, wyróżnić można następujące problemy napotykane podczas implementacji metod uczenia maszynowego:

- Wymagania wydajnościowe – są one ściśle powiązane ze złożonością obliczeniową wykorzystanych metod, wydajnością zastosowanego języka i wydajnością zastosowanej platformy sprzętowej. Docelowym efektem jest minimalizacja czasu wymaganego na uczenie modelu (chodź tutaj tolerowane są także długie czasy, szczególnie w przypadku dużych zestawów danch uczących) i czasu propagacji modelu (w przypadku czego minimalizacja czasu propagacji stanowi priorytet).

- Wymagania pamięciowe – wynikają one z wykorzystywanych platform sprzętowych i ich ograniczeń pamięciowych. Przykładem powyższego dylematu jest zastosowanie modeli uczenia maszynowego na platformach mobilnych i platformach systemów wbudowanych, gdzie obecne rozmiary pamięci RAM i pamięci masowej (szczególnie w przypadku platform wbudowanych) potrafią być wyraźnie ograniczone w stosunku do systemów komputerowych.

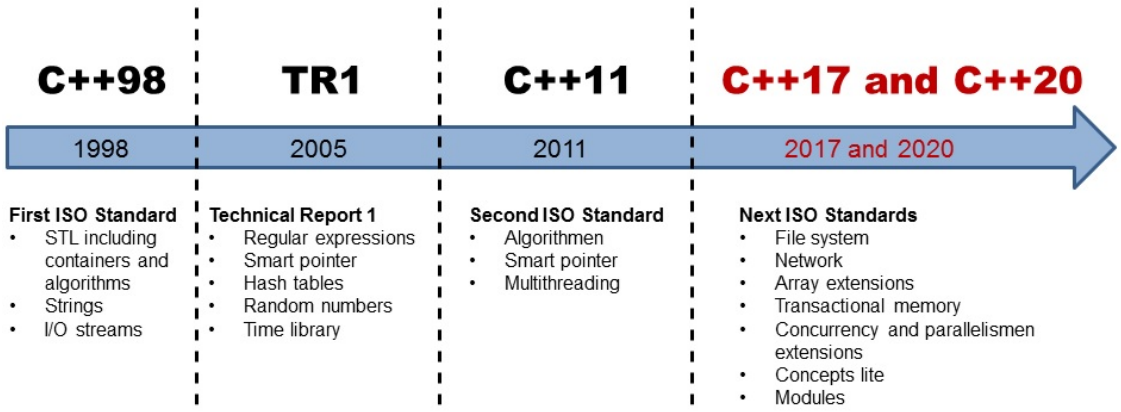
W trakcie rozwoju technologii uczenia maszynowego, postawiono stanowcze kroki w kierunku rozwiązywania powyższych problemów, aby sprostać narastającym wymaganiom związanym z coraz to nowymi i bardziej skomplikowanymi zastosowaniami sztucznej inteligencji. Dokonywano tego poprzez między innymi optymalizację algorytmów, dobór platform sprzętowych o wysokim taktowaniu, możliwym zrównolegleniu operacji, oraz wykorzystaniu wysoko wydajnych języków programowania, w szczególności języków mających możliwość wykorzystania wsparcia ze strony niskopoziomowych operacji.

2.2. Język C++ jako narzędzie do rozwiązania problemów uczenia maszynowego

Dostępne są różne języki i środowiska wspierające uczenie maszynowe, począwszy od języków takich jak Python, C++, Java czy Matlab. Jednak spośród wymienionych kandydatów szczególnie istotnym wyborem jest język C++.

C++ to język imperatywny charakteryzujący się silnym typowaniem, łączący programowanie niskopoziomowe dla konkretnych architektur z wysokopoziomowym programowaniem, w związku z czym oferuje programistom dużą kontrolę nad wykorzystaniem pamięci i możliwość optymalizacji w postaci m.in. dostosowywania wykorzystanych typów danych do wymagań funkcjonalnych tworzonej sieci, kontroli lokalizacji zmiennych (programista decyduje czy zmienna lub struktura znajdzie się na stosie czy stercie) oraz optymalizację czasów wywołań funkcji poprzez sugerowanie kompilatorowi utworzenia funkcji inline. W przeciwieństwie do języków skryptowych których kod jest interpretowany w trakcie wykonywania, takich jak Python i język środowiska Matlab, C++ jest językiem kompilowanym. Oznacza to, że program napisany w C++ przetwarzany jest z postaci tekstu do wykonawczego kodu binarnego dostosowanego do wybranej architektury procesora. Usuwa to całkowicie nadmiar złożoności obliczeniowej wykonywanego programu związanej z interpretacją poleceń i tłumaczeniem ich na język procesora danej platformy w trakcie wykonywania programu, gdyż jest to wykonywane tylko raz, na etapie kompilacji, dodatkowo pozwalając na zastosowanie przez kompilator mechanizmów optymalizacji dostępnych dla wybranej platformy.

Część mechanizmów z języka C++, wywodzących się jeszcze z języka C, pozwala na wykorzystanie wstawek kodu źródłowego w języku Assembler dla wybranego procesora, co zwiększa wydajność programu kosztem przenośności kodu. Dodatkowo niektóre platformy oferują API modułów akceleracji sprzętowej (jak np. system Android udostępniający *Neural Networks API, NNA* dla sieci neuronowych), co oferuje dodatkowe przyspieszenie czasu działania programu.



Rysunek 2.2. Multithreading in modern C++ - Modernes C++

Jedną z popularnych technik mających na celu znaczne zwiększenie wydajności modeli sztucznej inteligencji jest zrównoleglenie przetwarzania. Dostępność mechanizmów wielowątkowych dla procesorów (wprowadzonych w standardzie C++11 i

dalej rozwijanych, jak przedstawiono na rys. 2.2), oraz kompatybilność języka C++ z językiem CUDA pozwala wykonywać wiele obliczeń równolegle poprzez wykorzystanie wielu rdzeni lub oddelegowaniu części przetwarzania do karty (lub wielu kart) graficznej (gdzie ilość procesorów GPU znacząco przewyższa ilość rdzeni CPU). Dodatkowym atutem wykorzystania języka C++ przy tworzeniu modelu sztucznej inteligencji jest łatwa integracja z programami dedykowanymi do wysokiej wydajności, napisanymi w tym języku.

Wymienione wyżej mechanizmy i cechy charakterystyczne języka umożliwiają programistom znaczną optymalizację przygotowywanych rozwiązań sztucznej inteligencji, co przekłada się na bardziej efektywne zużycie pamięci, zabezpieczenie przed przeładowaniem stosu procesora, oraz krótsze czasy propagacji utworzonych modeli.

2.3. Cel powstania bibliotek

Implementacja mechanizmów pozwalających na tworzenie rozwiązań sztucznej inteligencji, z racji na swoją złożoność, wymagania dotyczące kompetencji twórców oraz konieczność optymalizacji jest czasochłonna i kosztowna. Tu z pomocą przychodzą biblioteki utworzone przez korporacje oraz społeczność programistów *open source*. Stanowią one gotowe zbiory mechanizmów (najczęściej pisane w sposób obiektowy, a więc ubrane w klasy posiadające określone zestawy metod), które są na bieżąco optymalizowane przez grupy programistów wykorzystujące je w prywatnych projektach lub pracy zawodowej. Oferują one możliwość wykorzystania gotowych modeli utworzonych w innych technologiach, a czasem także bezpośrednie przygotowanie modelu na podstawie odpowiednio sformatowanego i odpowiednio przystosowanego zestawu danych.

Użycie gotowych bibliotek nie tylko oszczędza koszta i przyspiesza tworzenie pożądanego rozwiązania sztucznej inteligencji, lecz także zapewnia większą niezawodność, gdyż elementy zawarte w bibliotece są implementowane, dokładnie testowane i poprawiane przez programistów o wysokich kompetencjach, jak m.in. w przypadku biblioteki TensorFlow posiadającej wsparcie od pracowników Google.

Więszkość bibliotek przeznaczonych do uczenia maszynowego, nawet wykorzystywanych w językach takich jak Python, napisana jest w języku C++, oferując API dostępne dla określonych języków docelowych. Niestety nie wszystkie biblioteki napisane w ten sposób oferują dostęp do całego API w języku C++ dla wykorzystujących je programów zewnętrznych, lub bywa on utrudniony i skomplikowany, co sprawia że w powszechnej praktyce część bibliotek dedykowanych dla języka C++ operuje na modelach przygotowanych w ramach innej, lub czasem nawet tej samej biblioteki, napisanch w innym języku. Częstym przypadkiem jest tutaj wykorzystanie właśnie języka Python do utworzenia grafu modelu lub modelu w formacie ONNX (ang. *Open Neural Network Exchange*).

W ramach analizy porównawczej w niniejszej pracy, porównywane będą biblioteki oferujące zarówno tworzeie modeli w ramach języka C++, jak i wymagające wykorzystania modeli z innego źródła.

Rozdział 3

Inżynieria danych eksperymentalnych i testowe szablony modeli

3.1. Omówienie danych eksperymentalnych

W celu zestawienia funkcjonalnego bibliotek uczenia maszynowego w języku C++ i przedstawienia przykładów konieczne było wybranie danych eksperymentalnych możliwych do wykorzystania jako porównawczy punkt odniesienia. Jako w/w dane wybrano bazę dotyczącą diagnostyki raka piersi „*Wisconsin Diagnostic Breast Cancer*" z listopada 1995 roku, w której zamieszczono wyniki obrazowania określone w sposób liczbowy. Autorami zestawu są Dr. Wiliam H. Wolberg, W. Nick Street oraz Olvi L. Mangasarian z Uniwersytetu Wisconsin [1]. Baza ta jest dostępna do pobrania z repozytorium Uniwersytetu Californii [2]. Dane mają następującą strukturę:

1) ID - numer identyfikacyjny pacjentki;
2) Diagnosis [*Malignant - M / Benign - B*] - charakter nowotworu, **zmienna odpowiedzi**;
3) Dane klasyfikujące:
    a) *Radius* - średnica guza;
    b) *Texture* - tekstura guza;
    c) *Perimeter* - obwód guza;
    d) *Area* - pole guza;
    e) *Smoothness* - gładkość, miara lokalnych różnic w promieniu guza;
    f) *Compactness* - zwartość, wykorzystywana do oceny stadium guza;
    g) *Concavity* - stopień wklęsłości miejsc guza;
    h) *Concave points* - punkty wklęsłości guza;
    i) *Symmetry* - symetria guza, pomagająca w ocenie charakteru przyrostu guza.

j) *Fractal dimention („coastline approximation" - 1)* - wymiar fraktalny pozwalający na ilościowy opis złożoności komórek nerwowych, umożliwiający stwierdzenie nowotworzenia się zbioru komórek.

Dla każdej ze zmiennych odpowiedzi została zebrana średnia wartość, odcyhelenie standardowe oraz średnia trzech największych pomiarów, gdzie każdy zestaw ustawiony jest sekwencyjnie (np. kolumna 3 - średni promień, kolumna 12 - odchylenie standardowe promienia, kolumna 22 - średnia trzech największych pomiarów promienia). Każda ze zmiennych ma charakter ciągły. Zredukowany zestaw danych, zawierający jedynie zmienne decyzyjne informujące o średnich wartościach znaleźć można jako dodatek do książki „*Biostatistics Using JMP: A Practical Guide*" autorstwa Trevora Bihla [3].

3.2. Charakterystyka i przetwarzenie danych

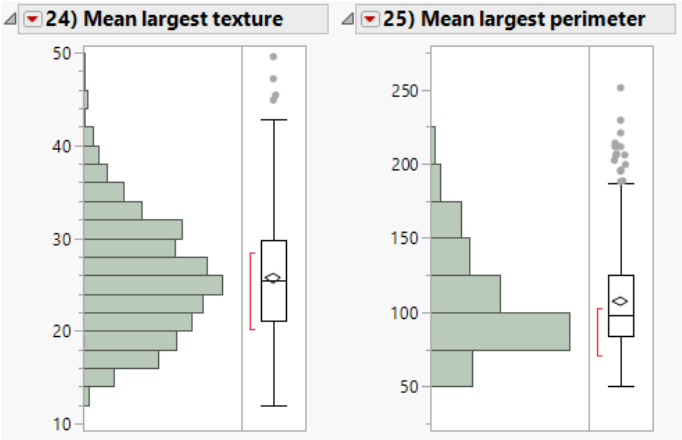
W celu przeprowadzenia procesu uczenia maszynowego, jednym z najistotniejszych kroków jakie należy podjąć jest wstępne zaznajomienie się z zestawem danych i jego analiza pod kątem rozkładu poszczególnych zmiennych oraz prawdopodobieństw. W tym celu wykorzystane zostało oprogramowanie JMP.

3.2.1. Analiza rozkładu danych

Proces analizy rozkładu rozpoczęty został od przyjrzenia się zmiennej odpowiedzi (*Diagnosis*). Rysunek 3.1 przedstawia uzyskany histogram, wraz z tabelą określającą ilość obserwacji danej klasy i współczynnik prawdopodobieństwa przynależności odpowiedzi do danej klasy. Zauważyć można, że dla użytego zestawu danych ilość zarejestrowano 357 obserwacji łagodnego raka piersi, a jego prawdopodobieństwo przynależności do klasy *Benign* wynosi $\approx$ 62,7%, natomiast do klasy *Malignant* przynależało 212 obserwacji z prawodpodobieństwem $\approx$ 37,3%.

Podczas analizy histogramów zmiennych decyzyjnych, stwierdzono że znaczna ilość ma charakter prawostronnie skośny oraz występują dla nich obserwacje odstające, o czym informuje znajdujący się po prawej stronie histogramu wykres okienkowy (ang. *box graph*), co przedstawiono na rysunku 3.2. Wyjątkiem okazała się zmienna *Mean Largest Concave Points*), która mimo lekkiej skośności, okazała się nie posiadać obserwacji odstających. Na podstawie tych informacji stwierdzono, że aby przygotować dane w odpowiedni sposób do procesu uczenia należy przeprowadzić ich czyszczenie oraz normalizację rozkładu.

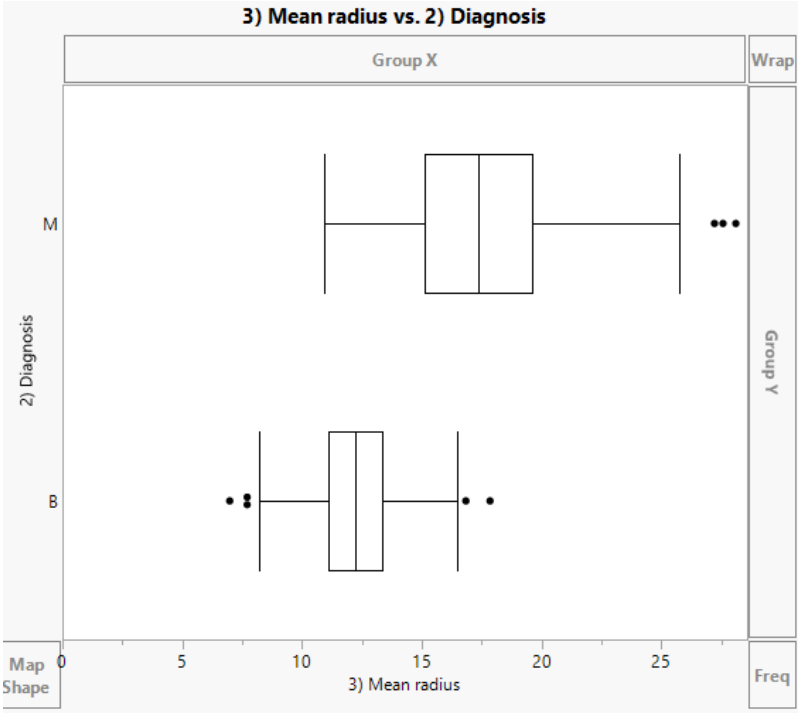
Rysunek 3.1. Histogram rozkładu zmiennej odpowiedzi



Rysunek 3.2. Przykłady histogramów zmiennych decyzyjnych

3.2.2. Czyszczenie i normalizacja rozkładu danych

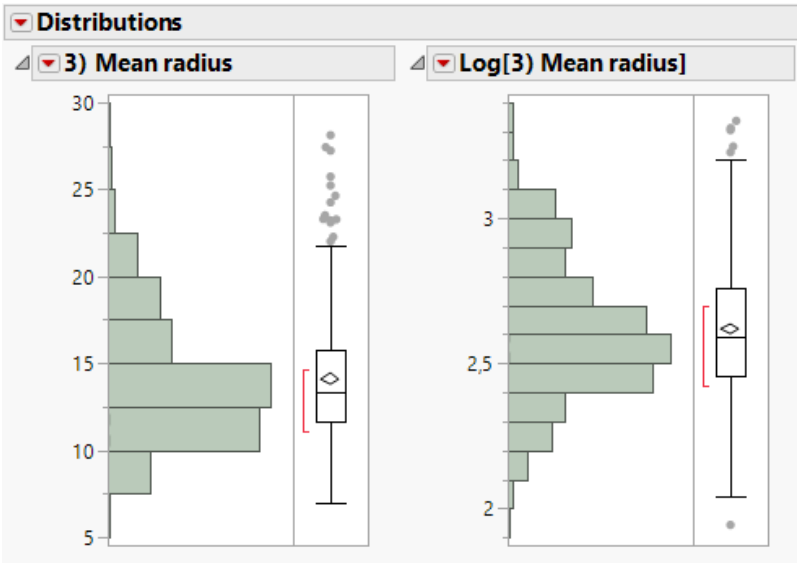
Na pełny zestaw danych składa się 569 obserwacji. Podczas wstępnej analizy stwierdzono istnienie 13 brakujących wartości dla regresora *Std err concave points*, dla których przyjęto wartość średnią z całej kolumny. Głównym problemem okazały się obserwacje odstające oraz skośności rozkładu. Do analizy obserwacji odstających wykorzystano wykresy okienkowe, gdzie oś Y reprezentowała zmienną odpowiedzi, natomiast oś X czyszczoną zmienną decyzyjną. Przykładowy wykres został przedstawiony na rysunku 3.3. Ze względu na bardzo małą ilość obserwacji zdecydowano się rozpocząć proces przystosowywania danych do uczenia poprzez normalizację ich rozkładu, aby zminimalizować lub wyeliminować konieczność usunięcia danych odstających.



Rysunek 3.3. Przykład analizy obserwacji odstających dla poszczególnych klas zmiennej odpowiedzi

W pierwszym podejściu zdecydowano się na zastosowanie transformacji logarytmicznej dla wszystkich zmiennych decyzyjnych i porównanie charakterystyk uzyskanych rozkładów z oryginalnymi. Zmienna *Mean largest concave points* okazała się posiadać rozkład bardzo zbliżony do standardowego, w związku z czym wyłączono ją z dalszej analizy normalizacji. Przykładowe wyniki przedstawiono na rysunku 3.4. Transformacja ta okazała się skutecznym rozwiązaniem jedynie dla następujących zmiennych:

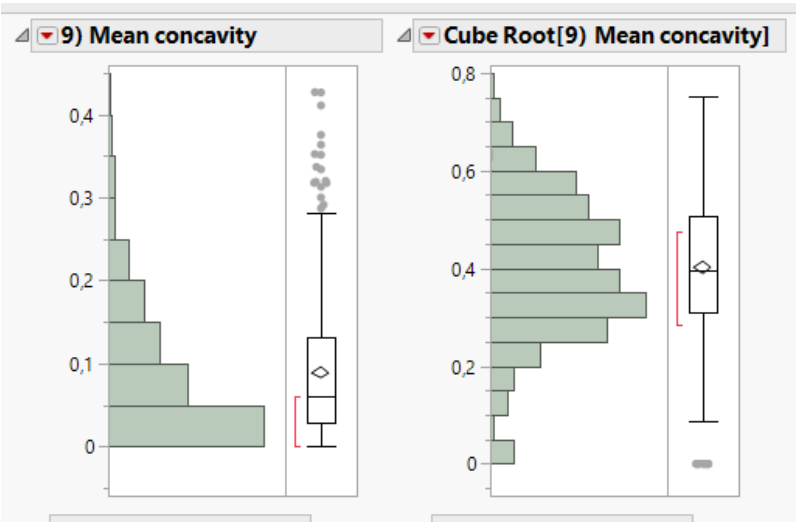
1. *Mean radius*;
2. *Mean texture*;
3. *Mean perimeter*,
4. *Mean area*;
5. *Mean smoothness*;
6. *Mean symmetry*;
7. *Std err texture*;
8. *Std err smoothness*;
9. *Std err compactness*;
10. *Std err concave points*;
11. *Mean largest texture*;
12. *Mean largest smoothness*;
13. *Mean largest compactness*.



Rysunek 3.4. Porównanie rozkładu danych przed i po transformacji logarytmicznej.

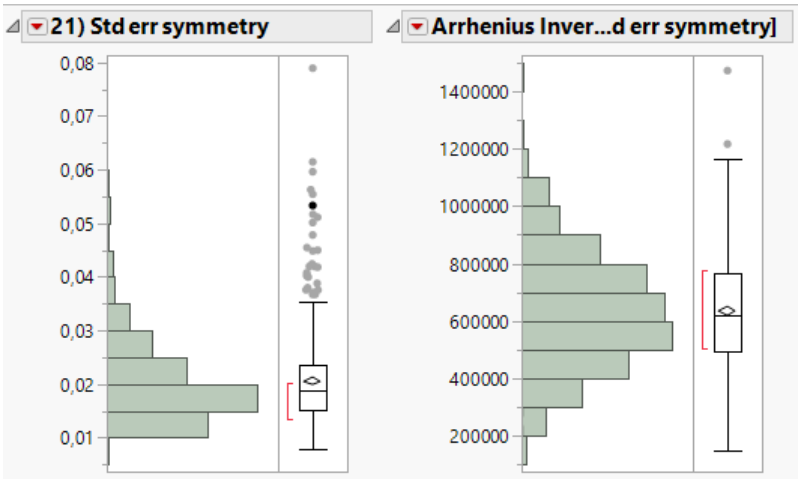
W drugim kroku podjęto próbę wykorzystania transformacji pierwiastkiem sześciennym dla pozostałych zmiennych decyzyjych, ze względu na jej skuteczność dla danych o rozkładzie prawoskośnym. Rysunek 3.5. przedstawia porównanie rozkładu zmiennej *Mean concavity* przed i po transfromacji pierwiastkiem sześciennym. Pomyślnie znormalizowano rozkład następujących zmiennych:

1. *Mean compactness*;
2. *Mean concavity*;
3. *Mean concave points*;
4. *Std err concavity*;
5. *Mean largest radius*;
6. *Mean largest perimeter*;
7. *Mean largest concavity*;
8. *Mean largest symmetry.*



Rysunek 3.5. Porównanie rozkładów danych przed i po zastosowaniu transformacji pierwiastkiem sześciennym.

Ostatecznym krokiem okazało się zastosownie odwrotnej transformacji Arrheniusa, Niestety część z uzyskanych zmodyfikowanych zmiennych decyzyjnych zachowała częściowy skośny rozkład, jednak inne przetestowane transformacje, jak m.in. pierwiastek kwadratowy, potęga kwadratowa, logarytm x+1, logarytm dziesiętny, funkcja potęgowa, funkcja wykładnicza, przyniosły rezultaty porównywalne lub gorsze od uzyskanego w wyniku w/w odwrotnej transformacji Arrheniusa. Rysunek 3.6 przedstawia porównanie uzyskanych rozkładów.



Rysunek 3.6. Porównanie uzyskanych rozkładów danych przed i po odwrotnej transformacji Arrheniusa.

Ze względu na bardzo małą ilość obserwacji, zdecydowano się na zachowanie wszystkich obserwacji odstających, aby zapobiec utracie informacji i zmianie uzyskanych w procesie normalizacji rozkładów.

3.3. Szablony docelowych modeli dla zadanych danych eksperymentalnych

Ze względu na dychotomiczny charakter zmiennej odpowiedzi, wybrany został przedstawiony poniżej zestaw metod dla których wykonano i przedstawiono testy praktyczne. Szablony struktury rozwiązań, takie jak np. wybór zmiennych uczestniczących w procesie uczenia, lub struktura sieci neuronowej zostały ustalone w sposób empiryczny z wykorzystaniem programu do uczenia maszynowego JMP.

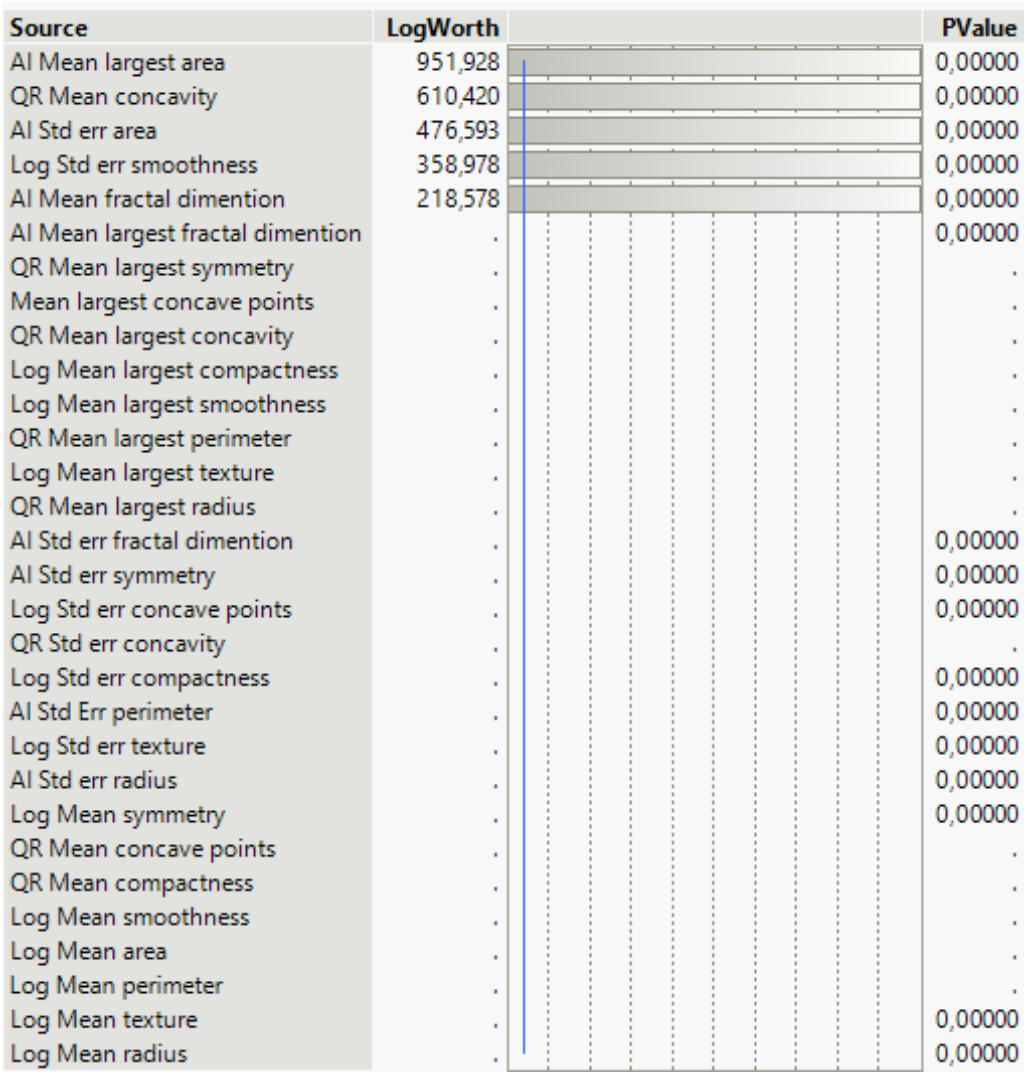
3.3.1. Regresja logistyczna

Badanie zależności w modelu regresji logistycznej odbyło się z wykorzystaniem wykresu wpływu zmiennej decyzyjnej na zmienną odpowiedzi opartego o p-wartość. Jako próg pozwalający na odrzucenie hipotezy zerowej (hipotezy o braku wypływu zmiennej na odpowiedź) przyjęto 0.05 jednostek. Rysunek 3.7 przedstawia w/w wykres wraz z p-wartościami dla poszczególnych zmiennych. Zauważyć można, że dla części zmiennych nie została wyznaczona p-wartość – oznacza to, że część zmiennych jest ze sobą skorelowanych.

Pierwszym krokiem w wybraniu istotnych zmiennych było usunięcie zmiennych skorelowanych, drugim natomiast stopniowe usuwanie zmiennych o p-wartości powyżej określonego progu. Rysunek 3.8 przedstawia listę wraz z wykresem kolumnowym istotnych regresorów. Ich lista, wraz z odpowiadającymi im p-wartościami została umieszczona w tabeli 3.1.

Nazwa zmiennej	p-wartość
Log mean largest texture	0,00000
Log mean largest compactness	0,00000
Cube root mean largest symmetry	0,00001
Arrhenius inverse std err symmetry	0,00005
Arrhenius inverse std err radius	0,00018
Cube root mean concave points	0,00056
Cube root mean largest concavity	0,00069
Log std err texture	0,00252
Cube root mean largest perimeter	0,00526
Log mean smoothness	0,04867
Log mean radius	0,04884

Tabela 3.1. Lista istotnych regresorów



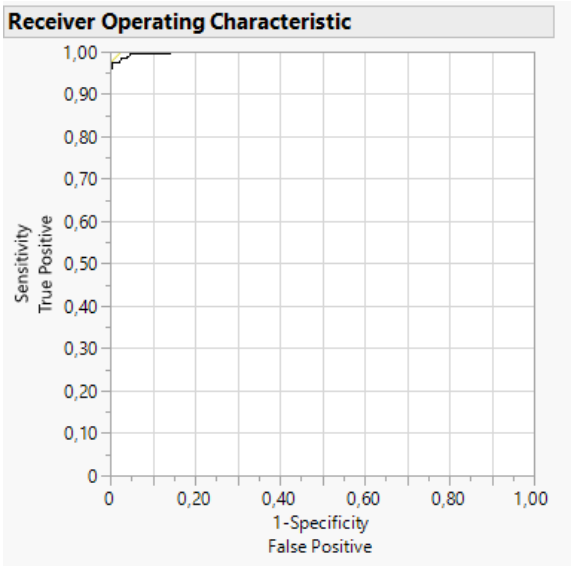
Rysunek 3.7. Wykres p-wartości dla całego zestawu zmiennych decyzyjnych.

Dla wybranego zestawu zmiennych model osiągnął dokładność na poziomie R^2 = 0.9401. Zgodnie z macierzą pomyłek, 207 obserwacji typu *Malignant* oraz 335 obserwacji *Benign* zostało zaklasyfikowanych poprawnie. Oznacza to, że model uzy-

skał tylko 2 wyniki typu *false-positive* (prawdopodobieństwo 0,6%) i 5 wyników typu *false-negative* (prawdopodobieństwo 2,4%) dla danych treningowych. Ze względu na mały zestaw obserwacji, ryzyko przeuczenia jest znikome, w związku z czym nie wytypowano zestawu danych walidacyjnych. Rysunek 3.9 przedstawia krzywą charakterystyczną odbiornika dla modelu.

Source	LogWorth	PValue
Log Mean largest texture	10,190	0,00000
Log Mean largest compactness	6,684	0,00000
QR Mean largest symmetry	4,873	0,00001
AI Std err symmetry	4,304	0,00005
AI Std err radius	3,748	0,00018
QR Mean concave points	3,255	0,00056
QR Mean largest concavity	3,159	0,00069
Log Std err texture	2,598	0,00252
QR Mean largest perimeter	2,279	0,00526
Log Mean smoothness	1,313	0,04867
Log Mean radius	1,311	0,04884

Rysunek 3.8. Wykres i p-wartości istotnych zmiennych decyzyjnych



Rysunek 3.9. Krzywa charakterystyczna odbiornika (ROC) dla modelu regresji logistycznej

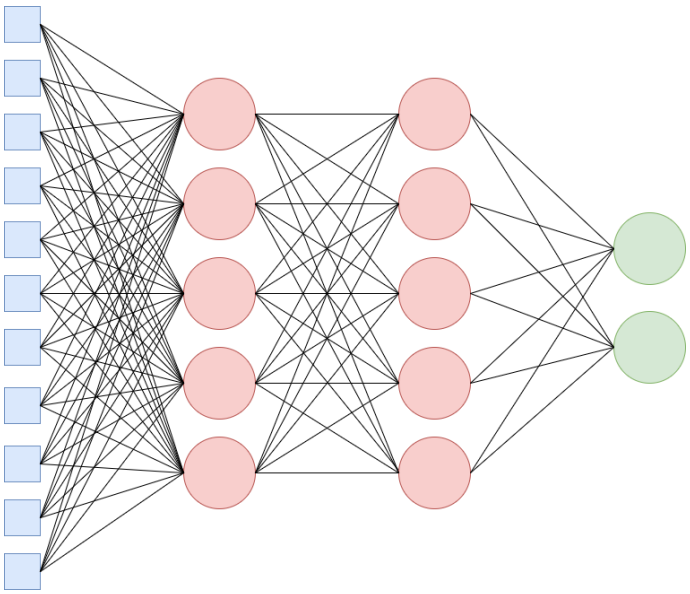
3.3.2. Głęboka sieć neuronowa

Do przygotowania sieci neuronowej wykorzystano zestaw zmiennych zawartych w tabeli 3.1. Dane zostały losowo podzielony na dane uczące i walidacyjne w propocji 80% do 20%. W wyniku prób i błędów, optymalny model uzyskano przy strukturze przedstawionej w tabeli 3.2. Graficzny schemat struktury został także przedstawiony na rysunku 3.10.

Środowisko JMP nie udostępnia informacji o funkcji aktywacji warstwy wyjściowej, w związku z czym w tabeli 3.2 została ona pominięta. Dla ziarna o wartości 1234 uzyskano model którego statystyka R^2 dla danych treningowych wyniosła 0.966268, natomiast dla danych testowych 0.9924547. Trafność dla losowo wybranego zestawu

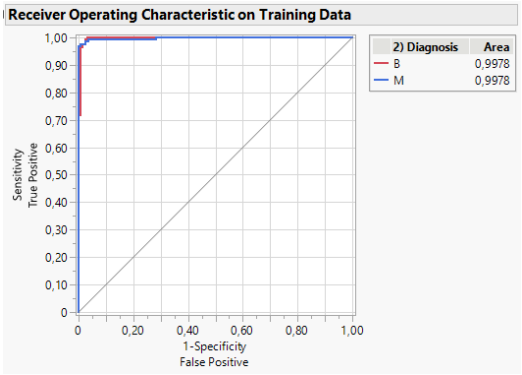
Typ warstwy	ilość neuronów	aktywacja
ukryta	5	tangens hiperboliczny
ukryta	5	tangens hiperboliczny
wyjściowa	2	—

Tabela 3.2. Struktura modelu sieci neuronowej

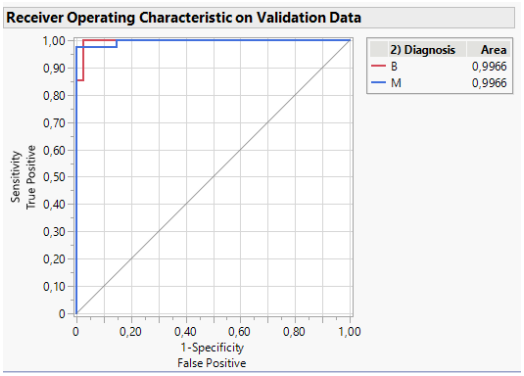


Rysunek 3.10. Schemat struktury sieci

testowego wyniosła 100%, natomiast dla danych uczących napotkano 5 przypadków *false-negative* (prawdopodobieństwo 3%) oraz 1 przypadek *false-positive* (prawdopodobieństwo 0,4%). Rysunki 3.11 oraz 3.12 przedstawiają krzywe charakterystyczne odbiornika dla zestawu testowego i walidacyjnego.



Rysunek 3.11. Krzywa charakterystyczna odbiornika dla zestawu testowego



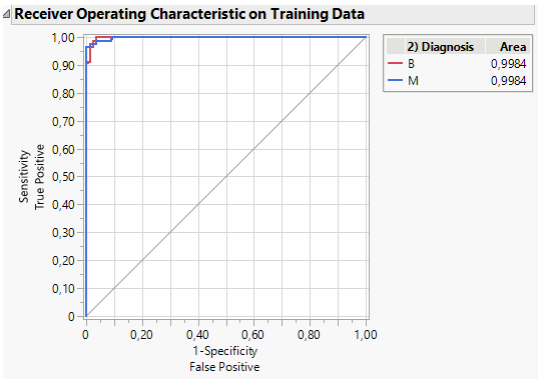
Rysunek 3.12. Krzywa charakterystyczna odbiornika dla danych walidacyjnych

3.3.3. Maszyna wektorów nośnych

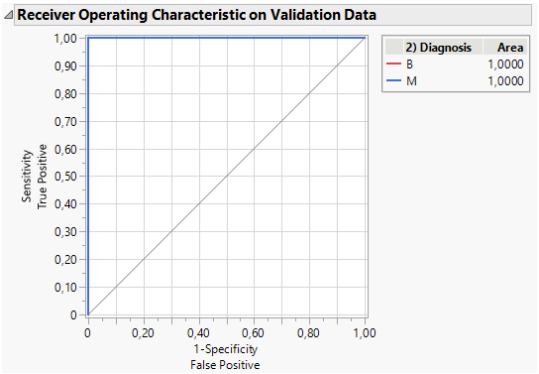
Do predykcji diagnozy wykorzystano ten sam zestaw regresorów, zawartych w tabeli 3.1. Ponownie w celu walidacji użyto metody wybrania losowego zestawu walidacyjnego spośród dostarczonych danych, w proporcji 80% obserwacji uczących i 20% testowych, z użyciem wartości 1234 dla ziarna generatora liczb pseudolosowych. Jako funkcję jądra maszyny wektorów nośnych (ang. Support Vector Machine, SVM) wybrano *Radial Basis Function*, która jest domyślnym wyborem dla SVM w środowisku JMP.

Zmienna decyzyjna	wartość X
Log mean largest texture	3,217
Log mean largest compactness	-1,5504
Cube root mean largest symmetry	0,65891
Arrhenius inverse std err symmetry	635100
Arrhenius inverse std err radius	38170
Cube root mean concave points	0,33665
Cube root mean largest concavity	0,5951
Log std err texture	0,1049
Cube root mean largest perimeter	4,7045
Log mean smoothness	-2,3502
Log mean radius	2,6191

Tabela 3.3. Wartości składowych X modelu dla poszczególnych zmiennych decyzyjnych



Rysunek 3.13. Krzywa charakterystyczna odbiornika dla danych uczących modelu SVM



Rysunek 3.14. Krzywa charakterystyczna odbiornika dla danych walidacyjnych modelu SVM

Utworzony w ten sposób model posiada generalizowaną statystykę R^2 na poziomie 0.97161 dla zestawu walidacyjnego, i uzyskał wskaźnik błędnej klasyfikacji wynoszący 0% dla danych testowych, oraz 1,3% dla danych uczących. Tabela 3.3 przedstawia wartości X dla poszczególnych regresorów. Rysunki 3.13 oraz 3.14 przedstawiają krzywe charakterystyczne odbiornika dla uzyskanego modelu.

Rozdział 4

Biblioteka TensorFlow

4.1. Wprowadzenie

Tnesorflow to darmowa biblioteka do uczenia maszynowego typu *open-source* stworzona przez Google. Oferuje ona możliwość przeprowadzenia płytkiego oraz głębokiego uczenia maszynowego, z wykorzystaniem własnych lub gotowych zestawów danych. Pozwala ona także na użycie pretrenowanych modeli jako rozwiązanie docelowe lub część składowa własnego modelu. Stanowi ona jedną z najpopularniejszych bibliotek ML, w szczególności wśród osób dopiero rozpoczynające swoje doświadczenia z tą gałęzią informatyki.

Jak podano na stronie domowej biblioteki Tensorflow[4], deklaruje ona bycie odpowiednią zarówno do przeprowadzania badań, jak i do wykorzystania w zastosowaniach przemysłowych. W momencie pisania niniejszej pracy, najnowszą dostępną wersją jest Tensorflow 2.11.

4.2. Metody wdrożenia modeli

Biblioteka umożliwia następujące metody wdrożenia modeli dla języka C++ [5][6][7]:

- API TensorFlow w języku C++
- Środowisko wykonawcze ONNX
- Zestaw narzędzi TensorFlow Lite
- Za pomocą Protobuf

4.2.1. API w języku C++

Z racji bycia napisanym w języku C++, biblioteka TensorFlow oferuje wsparcie do tworzenia, ternowania, i wdrażania modeli uczenia maszynowego w tym języku, poprzez udostępnienie użytkownikom interfejsu programowania aplikacji (ang. *Application Programming Interface, API*). Udostępnia ono operacje pozwalające manipulacje tensorami, wykonywanie operacji matematycznych na tensorach i ich elementach, budowę modelu w postaci grafu obliczeniowego, oraz jego uruchomienia

wykorzystując różne metody obliczeniowe, np. Adagrad oraz metodę spadku gradientowego [8]. Zastosowanie tego API pozwala na utworzenie i pracę z modelem bezpośrednio w aplikacji pisanej w języku C++.

Listing 4.1. Przykład utworzenia grafu obliczeniowego w TensorFlow realizującego konkatenację ciągów znakowych [9]

```
#include <tensorflow/cc/client/client_session.h>
#include <tensorflow/cc/ops/standard_ops.h>
#include <tensorflow/core/framework/tensor.h>

int main(int argc, char **argv) {
  using namespace tensorflow;
  using namespace tensorflow::ops;

  // create a root scope
  auto scope = Scope::NewRootScope();

  // define various constans/inputs on which we
  // will perform an operation
  auto hello = Const(scope, std::string("hello"));
  auto space = Const(scope, std::string("␣"));
  auto world = Const(scope, std::string("world␣!"));

  // StringJoin operation
  auto joinOp = StringJoin(scope, {hello, space, world});

  // create a session that takes our
  // scope as the root scope
  ClientSession session(scope);

  // Run
  std::vector<Tensor> outputs;
  TF_CHECK_OK(session.Run({joinOp}, &outputs));

  // See our output using DebugString that tells
  // more information about the tensor
  std::cout << "DebugString␣->␣" << outputs[0].DebugString() << std::endl;

  // we can also get the underlying data by calling flat
  std::cout << "Underlying␣Scalar␣value␣->␣" << outputs[0].flat<std::string>()
            << std::endl;

  return 0;
}
```

4.2.2. Środowisko wykonawcze ONNX

Open Neural Networks Exchange, (ONNX) to środowisko mające na celu standaryzację reprezentacji modeli uczenia maszynowego, autorstwa firmy Microsoft [6]. Pozwala ono na uniezależnienie modelu na etapie wdrożenia od biblioteki i języka w którym został stworzony, pozwalając na ich użycie przy pomocy pojedynczego mechianizmu. Wykorzystanie tej metody wdrażania modeli polega na przygotowaniu ich za pomocą API języka okraz eksport do struktury akceptowanej przez środowisko wykonawcze ONNX.

4.2.3. Zestaw narzędzi TensorFlow Lite

Stanowi zbiór narzędzi (ang. *toolbox*) umożliwiających uruchomienie przygotowanych modeli na platformach mobilnych oraz urządzeniach brzegowych (takich jak moduły IoT, lub ogólnie rozumiane systemy mokroprocesorowe). Pozwala on dodatkowo m.in. na wykorzystanie akceleracji sprzętowej oraz optymalizacji modelu. Przygotowywanie modeli do uruchomienia za pomocą tej metody na wybranej platformie składa się m.in. z przygotowania modelu z użyciem biblioteki TensorFlow Lite Model Maker, wykorzystania jednego z gotowych, pretrenowanych modeli, lub konwersji modelu TensorFlow z formatu protobuf [7].

4.2.4. Za pomocą Protobuf

Utworzony w bibliotece TensorFlow model jest zapisywany do formatu wykorzystywanego przez mechanizm Protocol Buffers autorstwa Google, zwanego potocznie *Protobuf*. Pozwala on na przechowywanie grafu obliczeniowego modelu w notacji niezależnej od języków programowania, z której następnie narzędzie Protobuf umożliwia wygenerowanie odpowiednich klas i struktur danych dla wybranego języka, jak np. C++ czy Python [10]. Pliki Protobuf mogą mieć formę tekstową, która ułatwia edycję i analizę człowiekowi, lub formę binarną, pozwalającą na znaczne zaoszczędzenie miejsca w przypadku przechowywania danych numerycznych, jak np. wagi połączeń sieci.

Listing 4.2. Konstruktor klasy *Model* odczytujący strukturę modelu z pliku Protobuf [11]

```
#include <string>

#include "third_party/tensorflow/core/framework/graph.proto.h"
#include "third_party/tensorflow/core/framework/tensor.h"
#include "third_party/tensorflow/core/lib/io/path.h"
#include "third_party/tensorflow/core/platform/env.h"
#include "third_party/tensorflow/core/platform/init_main.h"
#include "third_party/tensorflow/core/platform/logging.h"
#include "third_party/tensorflow/core/platform/types.h"
#include "third_party/tensorflow/core/public/session.h"

Model(const std::string& graph_def_filename) {
    tensorflow::GraphDef graph_def;
    TF_CHECK_OK(tensorflow::ReadBinaryProto(tensorflow::Env::Default(),
                                            graph_def_filename, &graph_def));
    session_.reset(tensorflow::NewSession(tensorflow::SessionOptions()));
    TF_CHECK_OK(session_->Create(graph_def));
  }
```

4.3. Formaty źródeł danych

W języku C++ biblioteka TensorFlow wymaga podania danych w postaci obiektów klasy szablonowej *std::vector* z biblioteki STL języka [11], lub za pomocą własnych klas, np. *FeedDict*, jednak nie określa w jaki sposób muszą te dane być przechowywane. W związku z tym, możliwe jest wykorzystanie dowolnego formatu przechowywania danych do uczenia, pod warunkiem zapewnienia mechanizmu ich parsowania.

Typ przechowywany przez obiekty std::vector zależny jest od ilości regresorów, i może przyjmować postać typu prostego lub złożonego. Dane powinny być rozgraniczone przed procesem uczenia na dwa wektory, z których jeden przechowuje dane uczące, natomiast drugi dane walidacyjne.

Listing 4.3. Przykładowa funkcja realizująca krok uczenia [11]

```
1 void RunTrainStep(const std::vector<float>& input_batch,
2                    const std::vector<float>& target_batch) {
3     TF_CHECK_OK(session_->Run({{"input", MakeTensor(input_batch)},
4                                {"target", MakeTensor(target_batch)}},
5                               {}, {"train"}, nullptr));
6 }
```

4.4. Metody przetwarzania i eksploracji danych

API biblioteki TensorFlow dla języka C++ oferuje wiele operacji możliwych do wykonania na danych wykorzystywanych do uczenia. Operacje te można podzielić na manipulujące strukturą zapisanych danych (pozwalające na przetworzenie tensorów lub optymalizację pracy z danymi), oraz manipulujące wartościami regresorów (jak np. operacje matematyczne). Poniżej wymieniono wybrane, szczególnie istotne funkcjonalności przetwarzania i eksploracji danych, wyszczególnione w dokumentacji [8]:

4.4.1. Operacje strukturalne

Są to operacje mające na celu modyfikację struktury w jakich przechowywane są dane. Biblioteka oferuje szereg funkcjonalności mających na celu reformatowanie danych w celu optymalizacji wykonywania na nich obliczeń[8]. Należą do nich:

- **bitcast** - zmiana typu tensora bez kopiowania danych;
- **dekwantyzacja** - zamiana na typ zmiennoprzecinkowy z formatu stałoprzecinkowego lub całkowitoliczbowej notacji wykładniczej;
- **udawana kwantyzacja** - wykorzystywana do przygotowania danych aby móc z nich obliczyć gradient (dostępne jest kilka rodzajów udawanej kwantyzacji);
- **padding** - otoczenie wartości tensora nadmiarowymi wartościami w celu dostosowania wymiarów (możliwe uzupełnienie np. zerami, lub wartościami lustrzanymi);
- **stacking** - złącza N tensorów R-wymiarowych w jeden tensor R+1-wymiarowy;
- **dynamic stitching i partitioning** - złączanie danych z wielu tensorów w jeden lub rozdzielanie z jednego na wiele;
- **serializacja** - konwersja tensora na serializowany format dla Protobuf.

4.4.2. Eksploracja danych

W dokumentacji API TensorFlow dla C++ możliwe jest znalezienie wielu operacji pozwalających manipulować wartościami danych w celu wyliczenia pewnych cech, lub dokonania obliczeń potrzebnych w trakcie procesu uczenia[8]. Do najistotniejszych z nich należą:

- wyliczanie odcisku palca danych (ang. *fingerprint*);
- weryfikacja czy dane zawierają wartości NaN lub inf;
- wyodrębnienie fragmentów obrazów do warstwy „depth";
- obliczenie gradientu z danych po udawanej kwantyzacji;
- obliczanie rzadkiego (ang. *sparse*) gradientu dla danego akumulatora;
- obliczenie odwrotnej permutacji tensora;
- normalizacja za pomocą kwantyzacji;
- wybór kandydatów do ewaluacji funkcji wyjściowych (ang. *candidate sampling*);
- dostosowywanie nasycenia, kontrastu i kolorystyki obrazów;
- dekodowanie obrazów z różnych formatów (np. gif, bmp, jpg, bmp);
- dekodowanie danych z różnych źródeł (np. csv, json);
- generowanie ramek (ang. *bounding box*);
- zmiana rozmiaru obrazów;
- rekodowanie obrazu z kolorystyki RGB na HSV;
- zachłanna selekcja ramek;
- różnorodne operacje matematyczne;
- pooling i konwoolucję;
- normalizację z pomocą gradientów;
- różnorodne funkcje aktywacji i straty;
- operacje na ciągach znakowych;
- algorytmy treningowe, jak np. Adagrad, Adadelta, centered RMSProp.

4.4.3. Sterowanie przepływem obliczeń

Oprócz wymienionych wcześniej poleceń optymalizujących lub przetwarzających dane, biblioteka TensorFlow posiada szereg operacji zarządzających przepływem danych w grafie obliczeniowym reprezentującym model. Należą do nich:

- operacje na zakumulowanych gradientach;
- ustawianie barier;
- operacje na mapach, kolejkach i tablicach tensorów.

4.5. Modele uczenia maszynowego

Biblioteka TensorFlow z racji dostarczania operacji na tensorach i realizacji przetwarzania za pomocą grafu obliczeniowego, wspiera szereg modeli zarówno płytkiego jak i głębokiego uczenia maszynowego. Należą do nich [12]:

- regresja liniowa;
- regresja logistyczna;
- regresja Deminga;
- regresja grzbietowa i lasso;
- maszyna wektorów nośnych;
- jednokierunkowa sieć neuronowa (ang. *feedforward neural network*);
- splotowa sieć neuronowa;
- rekurencyjna sieć neuronowa;
- autoenkodery;
- wzmocnione uczenie maszynowe;

Powyższe modele mogą zostać wykorzystane do rozwiązywania wielu problemów, w tym także o charakterystyce liniowej czy klasyfikacji. Poniższe fragmenty omawiają sposób implementacji poszczególnych modeli, oraz przedstawiają praktyczne przykłady dla wybranych z nich.

4.5.1. Regresja liniowa

Regresja liniowa przy wykorzystaniu API TensorFlow wykonywana jest poprzez metodę macierzy odwrotnej. Problem określony jest wzorem:

$$Ax = b \tag{4.1}$$

gdzie A odpowiada za macierz wartości regresorów, x za wektor współczynników poszczególnych zmiennych, a b stanowi wektor wartości oczekiwanych. Rozwiązanie w postaci ogólnej określić można wzorem:

$$x = (A^T A)^{-1} A^T b \tag{4.2}$$

Operacje te wykonać można wykorzystując transpozycję oraz iloczyn tensorów, dostępny w sekcji matematycznej API TensorFlow. Pozostałe rodzaje regresji przedstawione w dalszych sekcjach, korzystają z wyżej opisanego mechanizmu.

4.5.2. Regresja logistyczna

Implementacja regresji logistycznej wykorzystuje proces regresji liniowej, jednak dokonuje na wyjściu normalizacji przewidywanego wyniku z użyciem funkcji sigmoidalnej, otrzymując prawdopodobieństwo przynależności. Obserwacja jest uznawana za należącą do sprawdzanej klasy jeżeli znormalizowana wartość przekroczy pewną zadaną wartość progową.

Implementację rozpoczęto od przygotowania mechanizmu parsowania pliku zawierającego dane eksperymentalne do postaci macierzy regresorów i wektora diagnozy. Wykorzystano w tym celu możliwość parsowania pliku CSV z dostępnego API biblioteki.

4.5.3. Regresja Deminga

Polega ona na minimalizowaniu całkowitego dystansu pomiędzy punktami pomiarowymi a uzyskaną prostą, zamiast minimalizacji dystansu pionowego. W celu jej uzyskania należy wykorzystać mechanizm regresji liniowej, modyfikując funkcję straty, w celu minimalizacji następującego wzoru:

$$d = \frac{y_0 - (ax_0 + b)}{\sqrt{m^2 + 1}} \tag{4.3}$$

gdzie (x_0, y_0) oznaczają punkt pomiarowy, a prosta uzyskiwana w wyniku regresji dana jest wzorem y = ax + b. Powyższy wzór ma zastosowanie dla pomiarów w przestrzeni dwuwymiarowej (jeden regresor i jedna wartość odpowiedzi). W przypadku większej liczby regresorów należy zastosować metodę operatą o ilocznym skalarny wektorów N+1 wymiarowych, gdzie N oznacza ilość regresorów, za pomocą 3 punktów (punktu dla którego liczymy odległość, czyli obserwacji [punkt P], oraz dwóch punktów leżących na prostej uzyskanej w wyniku regresji [punkty A i B]). Metoda ta przedstawiona jest następującymi wzorami [13]:

$$\overrightarrow{pa} = P - A \tag{4.4}$$

$$\overrightarrow{ba} = B - A \tag{4.5}$$

$$t = (\overrightarrow{pa} \cdot \overrightarrow{ba})/(\overrightarrow{ba} \cdot \overrightarrow{ba}) \tag{4.6}$$

$$d = |\overrightarrow{pa} - t * \overrightarrow{ba}| \tag{4.7}$$

Wzór 4.7 przedstawia wynikową formułę obliczania odległości, która musi być minimalizowana w ramach funkcji straty.

4.6. Metody analizy modeli

4.7. Dostępność dokumentacji i źródeł wiedzy

Niestety biblioteka TensorFlow posiada bardzo ograniczone zasoby umożliwiające pisanie z jej użyciem programów w języku C++. Dokumentacja okazuje się bardzo ograniczona i niekompletna. Dodatkowo niewielka liczba osób korzysta z tej biblioteki w języku C++, przez co ciężko znaleźć artykuły lub publikacje które prezentowałyby przykłady kodu źródłowego lub omawiały sposób pracy z udostępnionym API. Wiele dostępnych w API poleceń jak np. Placeholder, nie znalazło się w oficjalnej liście dostępnych funkcji i klas.

Rozdział 5

Biblioteka Shark

Rozdział 6

Biblioteka Caffe

Rozdział 7

Biblioteka PyTorch

Rozdział 8

Podsumowanie

8.1. Nakład pracy

8.2. Funkcjonalności

8.3. Dostępność źródeł wiedzy i dokumentacja

Bibliografia

[1] Olvi L. Mangasarian Dr William H. Wolberg W. Nick Street. *Wisconsin Diagnostic Breast Cancer (WDBC)*. 1995. URL: https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic).

[2] Dheeru Dua i Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[3] Trevor Bihl. *Biostatistics Using JMP: A Practical Guide*. Cary, NC: SAS Institute Inc., 2017.

[4] Google Brain. *TensorFlow*. 2023. URL: https://www.tensorflow.org/.

[5] Vitaly Bezhachev. *How to deploy Machine Learning models with TensorFlow. Part 1 - make your model ready for serving*. 2017.

[6] Nour Islam Mokhtari. *How to Deploy TensorFlow Models in C++ in 3 different ways*. 2022.

[7] Google Brain. *TensorFlow Lite*. 2023. URL: https://www.tensorflow.org/lite.

[8] Google. *TensorFlow C++ API Reference*. 2023. URL: https://www.tensorflow.org/api_docs/cc.

[9] ksachdeva. *TensorFlow C++ Examples*. 2018. URL: https://github.com/ksachdeva/tensorflow-cc-examples/blob/master/examples/1_Introduction/src/hello-world.cc.

[10] Google. *A Tool Developer's Guide to TensorFlow Model Files*. 2023. URL: https://chromium.googlesource.com/external/github.com/tensorflow/tensorflow/+/r0.10/tensorflow/g3doc/how_tos/tool_developers/index.md.

[11] asimshankar. *Training TensorFlow Models in C++*. 2022. URL: https://gist.github.com/asimshankar/5c96acd1280507940bad9083370fe8dc.

[12] Nick McClure. *TensorFlow Machine Learning Cookbook*. Packt Publishing, Luty 2017.

[13] comingstorm. *Distance from point to n-dimentional line*. 2013. URL: https://softwareengineering.stackexchange.com/questions/168572/distance-from-point-to-n-dimensional-line.