Uczenie Maszynowe

Algorytm do indukcji reguł IRep++

Projekt końcowy

Maciej Bogusławski 331362
Hubert Kaczyński 331386

1. Streszczenie założeń projektu wstępnego

Celem projektu była implementacja i analiza algorytmu IRep++ (Incremental Reduced Error Pruning++) służącego do indukcji reguł decyzyjnych. Algorytm ten stanowi rozwinięcie podstawowego algorytmu IRep, wprowadzając ulepszoną metodę estymacji błędu rzeczywistego oraz techniki przeciwdziałania przeuczeniu, przy jednoczesnym zachowaniu mniejszej złożoności niż RIPPER.

W ramach projektu zaplanowano porównanie trzech algorytmów: IRep++, IRep oraz RIPPER pod kątem metryk klasyfikacji (accuracy, precision, recall) oraz czasu działania. Dodatkowo przewidziano eksperymenty badające wpływ rozmiaru zbioru treningowego oraz stosunku wielkości zbiorów rosnącego i przycinającego na jakość klasyfikacji.

Do eksperymentów wybrano zbiór danych Breast Cancer Wisconsin (Diagnostic) zawierający 569 rekordów z 30 cechami opisującymi cechy fizyczne komórek nowotworowych, klasyfikowanych jako złośliwe lub łagodne.
Link do źródła danych:
https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic

Jako główne źródło informacji na temat algorytmu IRep++ została użyta praca znajdująca się pod poniższym adresem:
https://www.researchgate.net/publication/220907085_IREP_a_Faster_Rule_Learning_Algorithm

2. Opis środowiska i budowy projektu

Uruchomienie kodu źródłowego projektu

W głównym folderze projektowym znajduje się plik *requirements.txt*, który zawiera moduły wymagane do poprawnego uruchomienia kodu projektu. W celu skorzystania z nich można zastosować środowisko wirtualne venv:
Utworzenie środowiska w katalogu projektu:
python3 -m venv venv
Aktywacja środowiska:
source venv/bin/activate
Odtworzenie pożądanego środowiska z pliku *requirements.txt*:
pip install -r requirements.txt
W tak przygotowanym środowisku możliwe jest uruchomienie plików .py zawartych w projekcie.

Moduły zamieszczone w pliku *requirements.txt* to *matplotlib* służący do przedstawiania wyników eksperymentów projektu na wykresach, *numpy* oraz *pandas* służące do obróbki i pracy na danych. Moduł *wittgenstein* służy do wykorzystania gotowych algorytmów IRep i RIPPER w celu porównania ich działania z zaimplementowanym IRep++ w eksperymentach.

Pliki projektu

Plik *wdbc.csv* zawiera zbiór danych Breast Cancer Wisconsin (Diagnostic) zawierający 569 rekordów z 30 cechami opisującymi cechy fizyczne komórek nowotworowych, klasyfikowanych jako złośliwe lub łagodne. Cechy są przedstawione przez liczby rzeczywiste. Klasa rekordu jest opisana jako „M” jeśli nowotwór jest złośliwy (malignant) lub „B” jeśli

nowotwór jest łagodny (benign). Do tego każdy rekord ma swój identyfikator ID. Na jeden rekord przypada jedna linijka dokumentu.

Plik *IReppp.py* zawiera główny kod źródłowy zaimplementowanego algorytmu IRep++. Znajduje się w nim klasa *IrepppClassifier,* której metoda *fit* trenuje klasyfikator, a metoda *predict* służy dla przewidzenia etykiety dla danych wejściowych. Plik zawiera również stałą *COLUMN_NAMES,* która zawiera listę nazw wszystkich cech obecnych w danych. Służy jako pomoc do operowania na danych. Szczegóły implementacji znajdują się w sekcji „*3. Opis algorytmu IRep++”*.

Plik *demonstration.py* zawiera kod, który umożliwia jednokrotne uruchomienie algorytmu IRep++ na danych zawartych w pliku *wdbc.csv* z prezentacją wyników oraz metryk dokładności, precyzji i czułości (accuracy, precision, recall) w terminalu.

Plik *experiments_compare.py* zawiera kod, który przeprowadza eksperymenty porównujące wyniki dokładności, precyzji, czułości i czasu trwania dla zaimplementowanego algorytmu IRep++ z algorytmami IRep oraz RIPPER z biblioteki *wittgenstein*. Szczegóły implementacji znajdują się w sekcji „*4. Opis testów i eksperymentów*”.

Plik *experiments_study.py* zawiera kod, który przeprowadza eksperymenty porównujące wyniki dokładności dla zaimplementowanego algorytmu IRep++ z algorytmami IRep oraz RIPPER z biblioteki *wittgenstein* w wypadkach zastosowania różnych rozmiarów zbioru testowego, rosnącego i przycinającego. Szczegóły implementacji znajdują się w sekcji „*4. Opis testów i eksperymentów*”.

Folder ./plots zawiera obrazy z wykresami będącymi wynikami działania eksperymentów projektu.

Plik *tests.py* zawiera terminalowe testy jednostkowe metod i funkcji pomocniczych użytych w implementacji algorytmu IRep++ oraz eksperymentach i demonstracji. Wszystkie testy przechodzą pozytywnie potwierdzając poprawne działanie implementacji.

3. Opis zaimplementowanego algorytmu IRep++

Algorytm IRep++ stanowi rozszerzenie klasycznego algorytmu IRep, wprowadzając ulepszone mechanizmy oceny i przycinania reguł w celu redukcji zjawiska przeuczenia. Algorytm działa, budując zbiór reguł klasyfikacyjnych dla klasy pozytywnej (w tym przypadku nowotworów złośliwych – klasa M).

W zaimplementowanej klasie *IRepppClassifier* proces uczenia rozpoczyna się od metody *fit*, która formatuje dane wejściowe i wywołuje główną pętlę uczenia *_learn*. Pętla ta działa iteracyjnie, tworząc kolejne reguły do momentu osiągnięcia kryterium stopu - wystąpienia *max_bad_rules* czyli (domyślnie 5) kolejnych reguł, które zostały odrzucone jako niewystarczająco dobre.

Każda iteracja składa się z czterech kluczowych etapów. Najpierw dane treningowe są losowo dzielone przez metodę *_split* na zbiór rosnący (grow set) i przycinający (prune set) w proporcji 2:1, co stanowi ulepszenie względem podstawowego IRep. Następnie metoda *_grow_rule*

tworzy nową regułę, rozpoczynając od pustej reguły i iteracyjnie dodając do niej warunki (literały). W każdym kroku wybierany jest warunek przynoszący największy przyrost informacji FOIL, obliczany przez metodę *_foil_gain* według wzoru: p * ($\log_2(p/(p+n))$ - $\log_2(p_0/(p_0+n_0))$), gdzie p, n - liczba pozytywnych i negatywnych przykładów pokrytych przez regułę z nowym warunkiem, a p0, n0 - liczba pozytywnych i negatywnych przykładów pokrytych przez regułę przed dodaniem warunku. Proces wzrostu reguły kontynuowany jest do momentu, gdy reguła nie popełnia błędów na zbiorze rosnącym, co sprawdzane jest przez metodę *_no_errors_on_grow_set*.

Po utworzeniu reguły następuje etap przycinania realizowany przez metodę *_prune_rule*. Algorytm testuje kolejne wersje reguły powstałe przez usuwanie warunków od końca i wybiera wariant o najwyższej dokładności na zbiorze przycinającym, przy czym nigdy nie akceptuje pustej reguły. Kluczowym elementem IRep++ jest wykorzystanie miary precyzji (precision) podczas przycinania, obliczanej jako stosunek poprawnie sklasyfikowanych przykładów pozytywnych do wszystkich przykładów pokrytych przez regułę.

Ostatnim etapem jest ocena, czy reguła powinna zostać zachowana, realizowana przez metodę *_keep*. Reguła jest akceptowana, jeśli pokrywa więcej przykładów pozytywnych niż negatywnych na pełnym zbiorze treningowym. Jeśli reguła zostaje zaakceptowana, jest dodawana do zbioru reguł, a przykłady przez nią pokryte są usuwane ze zbioru treningowego metodą *_not_covered*. W przeciwnym razie zwiększany jest licznik odrzuconych reguł.

Podczas wyszukiwania najlepszego podziału dla danej cechy metoda *_find_split* sortuje wartości cechy i dla każdej pary sąsiednich różnych wartości testuje punkt podziału z operatorami < i >. Implementacja wykorzystuje pomocnicze metody *_rule_covers* i *_literal_covers* do sprawdzania pokrycia przykładów przez reguły i pojedyncze warunki. Predykcja dla nowych danych realizowana jest przez metodę *predict*, która stosuje zasadę dysjunkcji - przykład klasyfikowany jest jako pozytywny, jeśli jest pokryty przez przynajmniej jedną z nauczonych reguł.

Zaimplementowany algorytm wykorzystuje reprezentację reguł jako list słowników, gdzie każdy słownik opisuje pojedynczy warunek zawierający indeks cechy, operator porównania, wartość progową oraz przyrost informacji. Taka struktura umożliwia efektywną weryfikację pokrycia oraz czytelne wyświetlanie reguł przez metodę *print_rules*, która formatuje warunki z wykorzystaniem nazw cech zdefiniowanych w stałej *COLUMN_NAMES*.

4. Opis eksperymentów

Eksperymenty 1

Plik experiments_compare.py zawiera kod przeprowadzający porównanie trzech algorytmów indukcji reguł: zaimplementowanego IRep++, oraz algorytmów Irep i RIPPER z biblioteki wittgenstein. Celem tego eksperymentu jest ocena jakości klasyfikacji oraz efektywności czasowej poszczególnych metod na tym samym zbiorze danych.

Eksperyment został zaprojektowany z wykorzystaniem wielokrotnych powtórzeń w celu zapewnienia statystycznej wiarygodności wyników. Dla każdego algorytmu przeprowadzanych jest *n_runs = 10* niezależnych uruchomień, z których każde wykorzystuje losowy podział danych na zbiór treningowy (80%) i testowy (20%) realizowany przez funkcję

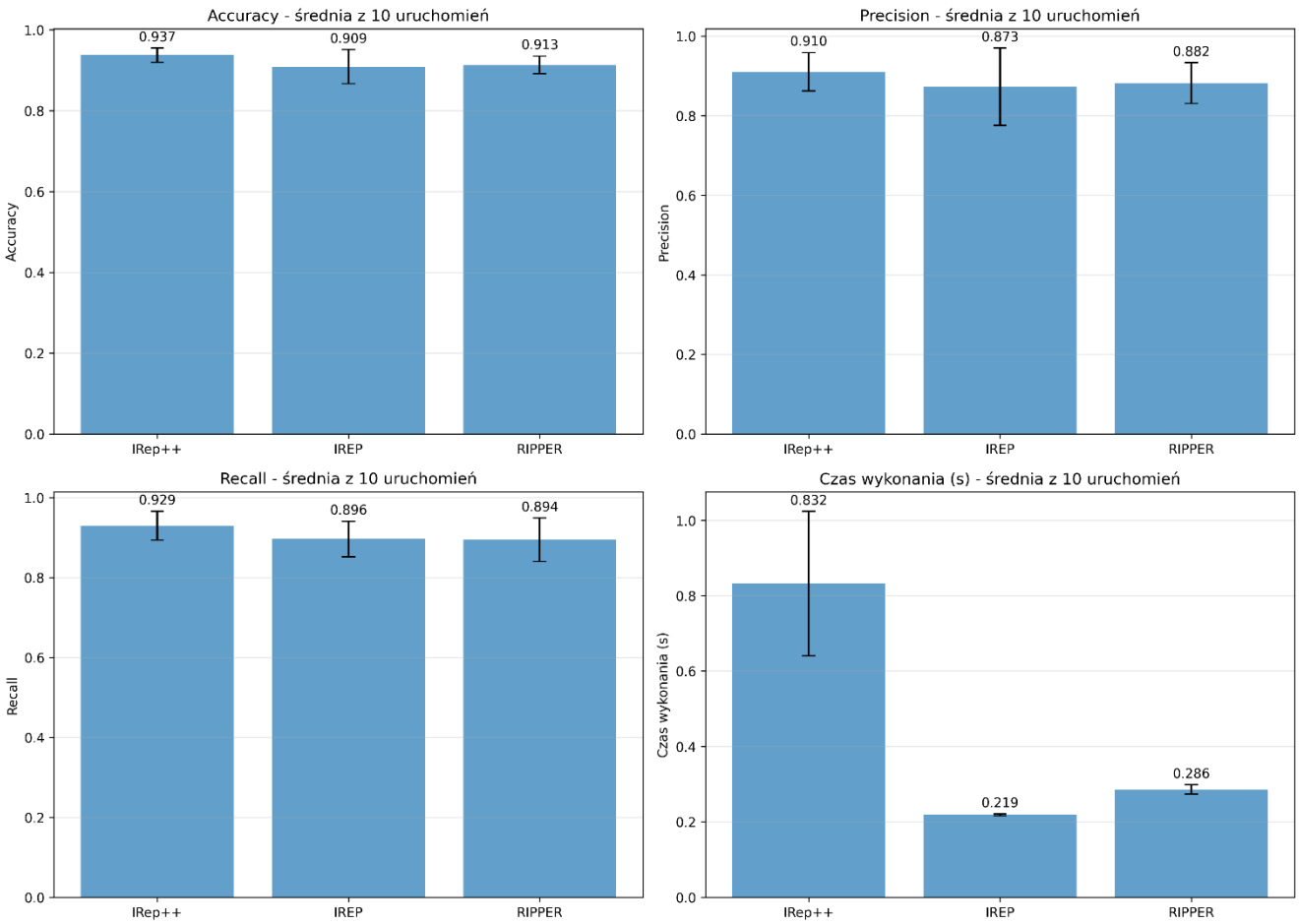
train_test_split. Takie podejście pozwala na uśrednienie wyników i obliczenie odchylenia standardowego, co umożliwia ocenę stabilności algorytmów.

W każdym uruchomieniu mierzone są cztery kluczowe metryki: accuracy (dokładność) - stosunek poprawnie sklasyfikowanych przykładów do wszystkich przykładów, precision (precyzja) - stosunek prawdziwie pozytywnych do wszystkich sklasyfikowanych jako pozytywne, recall (czułość) - stosunek prawdziwie pozytywnych do wszystkich rzeczywistych pozytywnych, oraz czas wykonania algorytmu mierzony za pomocą modułu *time*. Metryki te są obliczane przez funkcje pomocnicze *calculate_accuracy*, *calculate_precision* i *calculate_recall* zaimportowane z pliku *demonstration.py*.

Wszystkie wyniki są gromadzone w słowniku *results*, który dla każdego algorytmu przechowuje listy wartości czterech mierzonych metryk z kolejnych uruchomień. Po zakończeniu wszystkich eksperymentów następuje etap wizualizacji, w którym tworzony jest wykres złożony z czterech paneli (2×2) za pomocą biblioteki *matplotlib*. Każdy panel przedstawia średnią wartość jednej metryki dla trzech algorytmów w formie wykresu słupkowego, z paskami błędów reprezentującymi odchylenie standardowe oraz wartościami liczbowymi umieszczonymi nad słupkami. Gotowy wykres jest zapisywany do pliku *./plots/comparison_results.png*.

Dodatkowo program wyświetla w terminalu szczegółowe podsumowanie, prezentujące dla każdego algorytmu średnią wartość oraz odchylenie standardowe wszystkich mierzonych metryk w formacie *mean* ± *std*.

Wyniki przeprowadzonych eksperymentów:



Wnioski z przeprowadzonych eksperymentów
Analiza wyników z pliku *experiments_compare.py* wykazała, że zaimplementowany algorytm IRep++ osiągnął najlepsze wyniki pod względem wszystkich trzech mierzonych metryk jakości klasyfikacji - accuracy, precision i recall - przewyższając zarówno klasyczny Irep, jak i bardziej zaawansowany algorytm RIPPER. Ta przewaga IRep++ wskazuje na skuteczność zastosowanych mechanizmów ulepszających proces indukcji reguł, w szczególności ulepszonej metody przycinania wykorzystującej miarę precyzji oraz mechanizmów przeciwdziałania przeuczeniu.

Jednak wyższa jakość klasyfikacji wiąże się z istotnym kompromisem czasowym. Dla tej implementacji czas wykonania IRep++ okazał się być około trzykrotnie dłuższy w porównaniu do zaimportowanych z biblioteki *wittgenstein* algorytmów Irep i RIPPER. Co więcej, czas działania IRep++ charakteryzował się znacznie większym odchyleniem standardowym niż konkurencyjne algorytmy. Duże odchylenie standardowe czasu wykonania IRep++ sugeruje, że czas działania algorytmu jest silnie zależny od charakterystyki konkretnego losowego podziału danych. W niektórych uruchomieniach algorytm może szybko znaleźć dobre reguły i zakończyć działanie po niewielkiej liczbie iteracji, podczas gdy w innych przypadkach może potrzebować więcej iteracji i testowania większej liczby potencjalnych warunków.

Eksperymenty 2

Plik experiments_study.py zawiera kod przeprowadzający dwa eksperymenty mające na celu zbadanie wpływu kluczowych parametrów na jakość działania algorytmów IRep++, Irep i RIPPER. W przeciwieństwie do eksperymentów porównawczych, które oceniają ogólną wydajność algorytmów, te eksperymenty koncentrują się na analizie wrażliwości metod na zmiany w podziale danych na zbiory rosnące, przycinające, treningowe i testowe.

Wpływ rozmiaru zbioru treningowego
Pierwszy eksperyment bada, jak zmiana proporcji podziału danych na zbiór treningowy i testowy wpływa na dokładność klasyfikacji. Testowanych jest siedem różnych rozmiarów zbioru treningowego: od 30% do 90% całego zbioru danych (wartości *train_sizes = [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]*). Dla każdego rozmiaru przeprowadzanych jest 10 niezależnych uruchomień z losowym podziałem danych, co pozwala na obliczenie średniej dokładności oraz odchylenia standardowego dla każdego algorytmu.

Wpływ stosunku rozmiarów zbioru rosnącego i przycinającego
Drugi eksperyment koncentruje się na podziale danych treningowych na zbiór rosnący (grow set) i przycinający (prune set). Badanych jest siedem różnych wartości proporcji zbioru rosnącego: od 30% do 90% danych treningowych (wartości *grow_ratios = [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]*). W tym eksperymencie stały rozmiar zbioru testowego wynosi 20%, a zmienia się tylko wewnętrzny podział danych treningowych.

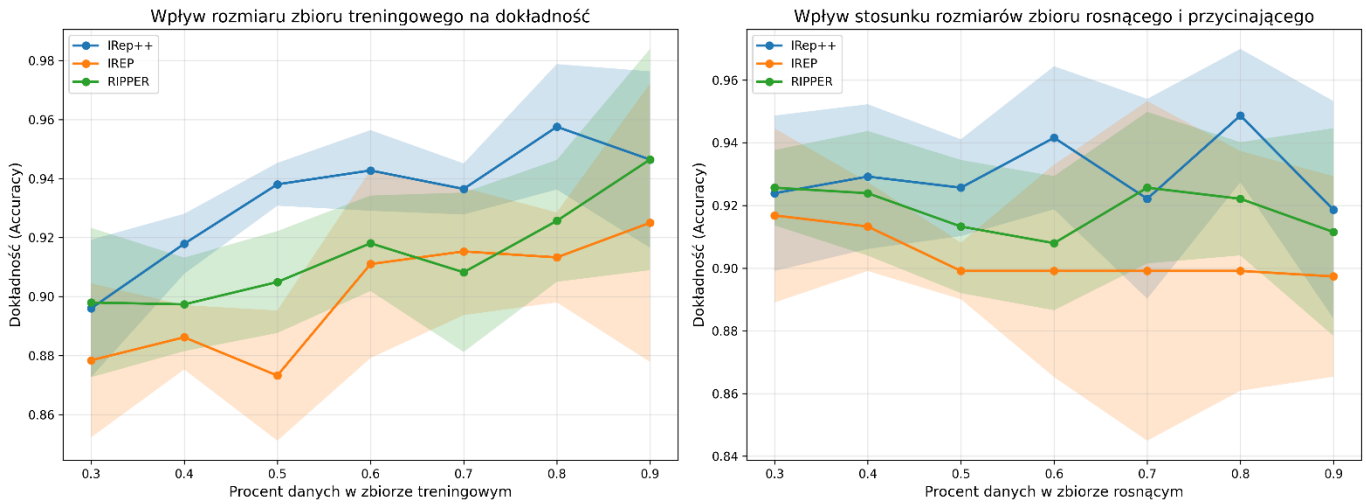
Dla zaimplementowanego IRep++ proporcja ta jest kontrolowana przez parametr grow_ratio przekazywany do konstruktora klasy *IRepppClassifier*. Dla algorytmów z biblioteki wittgenstein wykorzystywany jest odpowiadający mu parametr *prune_size*, który reprezentuje proporcję zbioru przycinającego (czyli *1 - grow_ratio*). Większy zbiór rosnący pozwala na uczenie bardziej szczegółowych reguł, ale mniejszy zbiór przycinający może być niewystarczający do efektywnego usuwania nadmiarowych warunków.

Wizualizacja i prezentacja wyników

Wyniki obu eksperymentów są prezentowane na dwóch panelach wykresów liniowych. Każdy wykres pokazuje zależność średniej dokładności od badanego parametru dla wszystkich trzech algorytmów, z zacieniowanymi obszarami reprezentującymi zakres odchylenia standardowego (*mean* ± *std*).

Program kończy się szczegółowym podsumowaniem w terminalu, prezentującym dla każdego algorytmu i każdej wartości testowanego parametru dokładną wartość średniej dokładności wraz z odchyleniem standardowym. Kompletny wykres zapisywany jest do pliku *./plots/study_results.png*, umożliwiając późniejszą analizę i porównanie wyników.

Wyniki przeprowadzonych eksperymentów:



Wnioski z przeprowadzonych eksperymentów

Pierwszy eksperyment z pliku experiments_study.py wykazał spodziewany trend - wszystkie trzy algorytmy osiągały wyższą dokładność wraz ze wzrostem proporcji danych przeznaczonych na zbiór treningowy. To naturalne potwierdza, że większa ilość danych treningowych pozwala algorytmom na lepsze nauczenie się wzorców charakteryzujących dane i budowę bardziej reprezentatywnych reguł klasyfikacyjnych. Dodatkowo Irep++ charakteryzował się na ogół lepszymi wynikami dokładności prawie dla każdej próbki, co jest pożądanym wynikiem – ma on być usprawnieniem w porównaniu z pozostałymi konkurentami.

Drugi eksperyment, badający wpływ proporcji między zbiorem rosnącym a przycinającym, ujawnił bardziej złożoną dynamikę. Wszystkie algorytmy wykazały zwiększone odchylenie standardowe przy większych wartościach parametru *grow_ratio* (większym zbiorze rosnącym). To zachowanie można wyjaśnić faktem, że przy dużym zbiorze rosnącym i małym zbiorze przycinającym algorytmy mają tendencję do tworzenia bardzo szczegółowych, specyficznych reguł na podstawie obszernego zbioru rosnącego, ale ograniczony zbiór przycinający może być niewystarczający do skutecznego usunięcia nadmiarowych warunków. W konsekwencji efektywność przycinania staje się bardziej zależna od szczególnych charakterystyk konkretnego losowego podziału, co prowadzi do większej wariancji wyników.

Widoczny jest też brak jasnej tendencji wzrostu lub spadku dokładności. Nie da się również jednoznacznie wskazać najlepszego algorytmu, ale Irep jest z pewnością najgorszy dla wszystkich próbek, co ponownie potwierdza to, że Irep++ i RIPPER są jego usprawnieniami.