Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Ewelina Karbowiak
Nr albumu: 332214

Analiza struktury i ekstrakcja wiedzy z klasyfikatorów otrzymanych metodą XGBoost

Praca magisterska
na kierunku MATEMATYKA

Praca wykonana pod kierunkiem
dra hab. Przemysława Biecka
Zakład Statystyki Matematycznej

Marzec 2019

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się 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 pracy

Streszczenie

Celem pracy magisterskiej było znalezienie sposobu identyfikacji interakcji w modelach XGBoost i LightGBM, a także stworzenie narzędzia, które pozwala badać ważność zmiennych i interakcji w danym modelu.

W pracy omówiono metody XGBoost i lightGBM, które są przykładami gradient boostingu drzew oraz pakiety w środowisku R, w których metody te zostały wykorzystane. Opisano również algorytmy optymalizacyjne użyte w tych pakietach, oraz najważniejsze parametry funkcji.

Następnie opisano miary ważności zmiennych oraz przeanalizowano metodykę wyjaśniania predykcji pojedynczej obserwacji. Wykonane zostały również symulacje w celu znalezienia interakcji w modelu.

W ramach pracy stworzono nowy pakiet `EIX`, którego zadaniem jest analiza struktury wyżej wymienionych modeli. Pakiet lokalizuje interakcje w modelu oraz posiada funkcję wizualizującą model. Oblicza także ważność interakcji i zmiennych zawartych w modelu za pomocą różnych miar – zarówno nowych, jak i tych dostępnych wcześniej. Dodatkowo daje też możliwość wyjaśnienia wpływu poszczególnych zmiennych na predykcję wybranej pojedynczej obserwacji.

W dalszej części pokazano przykład użycia pakietu na danych dotyczących odejścia pracowników z firmy i opisano możliwości stworzonego pakietu.

Słowa kluczowe

XGBoost, lightGBM, gradient boosting, drzewo, analiza struktury modelu, interakcja, ważność zmiennych, wizualizacja modelu, analiza predykcji obserwacji

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.2 Statystyka

Klasyfikacja tematyczna

68 Computer science
68Q Theory of computing
68Q32 Computational learning theory
62 Statistics
62P Applications
62P20 Applications to economics

Spis treści

Wprowadzenie

Modele uczenia maszynowego na podstawie zebranych danych pozwalają zrozumieć i opisać zjawiska zachodzące w rzeczywistym świecie, a także przewidywać przyszłe zdarzenia. Informacje jakie dostarczają przyczyniają się do rozwoju wielu dziedzin nauki. Pozwalają także np. optymalizować koszty i strategie dotyczące rozwoju przedsiębiorstw. Analiza danych oraz wnioskowanie na ich postawie znajduje więc zastosowanie w wielu dziedzinach - medycynie, przemyśle, sektorze finansowym, reklamie, czy w mediach społecznościowych.

Wraz ze wzrostem liczby danych dostarczanych do obróbki, rośnie złożoność modeli statystycznych. Jest to możliwe dzięki stale rosnącej mocy obliczeniowej komputerów. Wysoki poziom skomplikowania modeli bardzo często idzie w parze z dużą dokładnością. Problemem bywa jednak brak ich interpretowalności. Coraz częściej stosuje się tzw. czarne skrzynki - modele, które są na tyle złożone, że bezpośrednie prześledzenie ścieżki od danych do predykcji jest bardzo czasochłonne, a czasami niemożliwe. Przykładami takich modeli są lasy losowe, sieci neuronowe, czy omawiane w tej pracy modele związane z gradient boostingiem drzew. Są one zazwyczaj dokładniejsze niż tradycyjne, proste modele statystyczne, jednak przeciętny użytkownik nie jest w stanie ocenić, czy model korzysta z danych we właściwy sposób.

Zrozumienie jakie zmienne mają wpływ na wyniki modelu pozwala ocenić jego wiarygodność. Wiedza ta ma zastosowanie w wielu dziedzinach. Pozwala np.: na wdrożenie odpowiedniej startegii finansowej w sektorze bankowym, czy dobór optymalnej metody leczenia pacjenta ze schorzeniem immunologicznym. Wyjaśnienie modelu może być pomocne, w sytuacji gdy użytkownik stwierdza, że model nie korzysta z danych w oczekiwany sposób -zgodny z dostępną wiedzą i doświadczeniem.

Celem tej pracy magisterskiej było opracowanie metody rozpoznawania interakcji w komitetach drzew oraz analiza struktury modelu uwzględniająca interakcje. Został w niej opisany popularny model gradient boostingu drzew XGBoost i jego zoptymalizowana wersja LightGBM. Są to jedne z najdokładniejszych modeli używanych w dziedzinie uczenia maszynowego. Potwierdzają to, liczne wygrane konkursy na portalach takich jak: Analytics Vidhya i Kaggle. Niestety złożoność tych modeli nie pozwala na bezpośrednią ich interpretację i zrozumienie sposobu w jaki uzyskiwane są wyniki. W ramach pracy magisterskiej stworzyłam pakiet `EIX` *(Explain Interaction in Xgboost)*, którego celem jest wyjaśnienie modelu stworzonego metodą XGBoost ze szczególnym uwzględnieniem interakcji zachodzących w modelu.

Pakiet `EIX` jest narzędziem, które pozwala znaleźć interakcje występujące w modelu. Bada również ważność zmiennych i interakcji za pomocą różnych miar, z których część była już dostępna w pakiecie `xgboost`. Daje możliwość prześledzenia wpływu poszczególnych zmiennych na predykcję pojedynczej obserwacji z uwzględnieniem interakcji lub bez. Rozwiązanie to, za zgodą autora, korzysta z kodu zaimplementowanego w pakiecie xgboostExplainer. Oprócz tego prezentowany pakiet zawiera możliwości wizualizacji modelu, miar ważności zmiennych i interakcji, a także predykcji pojedynczej obserwacji. Powyższe funkcjonalności sprawiają, że narzędzie to jest przyjazne dla użytkownika i może być przydatne w zastosowaniach, gdzie aspekt wizualizacji jest istotny.

W pierwszym rozdziale został zdefiniowany model uczący się pod nadzorem, drzewo decyzyjne, jak również opisane zostały dwa rodzaje komitetów klasyfikatorów - bagging i boosting. Następnie opisany został model XGBoost i lightGBM oraz algorytmy optymalizacyjne zastosowane w pakietach o tych samych nazwach. W rozdziale drugim została omówiona metodologia pakietu `EIX`, a dokładnie miary ważności zmiennych, symulacje wykonywane w celu znalezienia strategii lokalizowania interakcji modelu oraz sposób wyjaśniania predykcji pojedynczej obserwacji w modelu. W trzecim rozdziale zostały dokładnie omówione funkcje zawarte w pakiecie `EIX`. W ostatnim rozdziale, przeprowadzono prezentację możliwości pakietu `EIX` na przykładzie danych dotyczących odejścia pracowników z firmy.

Rozdział 1

Modele XGBoost i LightGBM

W ramach tej pracy magisterskiej będą rozpatrywane modele uczenia się pod nadzorem. Są to modele, które posiadają wektor wejściowy (zmienne objaśniające) i wektor wyjściowy (zmienna objaśniana). Celem modelu jest znalezienie związku między zmiennymi objaśniającymi, a zmienną objaśnianą.

Niech zbiór $\mathbf{S}$ będzie n-elementową próbką treningową:

$$\mathbf{S} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\},$$

gdzie y_i jest zmienną objaśnianą, a $\mathbf{x}_i$ jest p-wymiarowym wektorem zmiennych objaśniających. Przez $\mathcal{S}$ oznaczono przestrzeń generowaną przez obserwacje. Przez $\mathbf{X}_i$ oznaczono zmienną losową, której realizacją jest obserwacja $\mathbf{x}_i$. Analogicznie Y_i to zmienna losowa, której wartością jest y_i. Zakładamy, że zmienna Y_i jest zależna od $\mathbf{X}_i$ i celem problemu uczenia maszynowego będzie znalezienie oczekiwanej wartości Y_i, co oznaczymy jako:

$$\mathbb{E}(Y_i|\mathbf{X}_i = \mathbf{x}_i) = f(\mathbf{x}_i),$$

gdzie $f(x)$ to funkcja $f : \mathbb{R}^p \to \mathcal{Y}$, która opisuje model, a zbiór $\mathcal{Y}$ to zbiór możliwych wyników modelu. Dla problemu regresji $\mathcal{Y} = \mathbb{R}$, dla klasyfikacji $\mathcal{Y} = \mathcal{G}$, gdzie zbiór $\mathcal{G}$ to skończony zbiór etykiet klas zmiennej Y i $|\mathcal{G}| = g$. Przez $\hat{y}$ została oznaczona predykcja modelu.

1.1. Drzewo decyzyjne

Drzewa decyzyjne po raz pierwszy prawdopodobnie pojawiły się w literaturze dotyczącej badań socjologicznych około 1963 roku. Po blisko 20 latach została wydana przez Breinman'a, Friedman'a, Ohlsen'a i Stone'a książka *Classification and Regression Trees*, która sprawiła, że drzewa decyzyjne zaczęły być powszechnie stosowane przy problemach uczenia maszynowego [28].

Drzewo, którego zadaniem jest klasyfikacja nazywamy drzewem decyzyjnym lub klasyfikacyjnym. Drzewo wykorzystywane do przybliżenia funkcji regresji to drzewo regresyjne.

Poniżej kilka definicji dotyczących drzew wykorzystywanych jako modele statystyczne:

Definicja 1.1.1 Drzewo decyzyjne lub drzewo regresyjne *to skierowany graf acykliczny i spójny, składający się z wierzchołków i krawędzi nazwanych* gałęziami. Korzeń (ang.*root*) *to wierzchołek początkowy drzewa decyzyjnego, który posiada wyłącznie krawędzi wychodzące.* Węzeł (ang.*node*) *to każdy wierzchołek drzewa, który posiada krawędź wychodzącą. Węzeł, z którego wychodzi krawędź nazywa się* rodzicem *wierzchołka, do którego krawędź ta jest skierowana.*

Wierzchołek ten nazwiemy dzieckiem. Liść (ang.*leaf*) *to wierzchołek grafu, który nie posiada żadnego dziecka.*

W środowisku R dostępnych jest wiele algorytmów i funkcji tworzących drzewa decyzyjne. Algorytmy różnią się sposobem podziału węzła. Schemat budowy drzewa jest natomiast podobny.

Przy tworzeniu nowego węzła, rozważane są wszystkie możliwe podziały na dwa rozłączne zbiory dla każdej z p zmiennych. Następnie dla każdego podziału sprawdza się miarę zróżnicowania zmiennej objaśnianej y w obu uzyskanych podzbiorach i wybiera ten podział który minimalizuje zróżnicowanie w węźle. W kolejnym kroku sprawdzany jest warunek podziału. Jeśli jest spełniony, dzieli się zbiór na dwa podzbiory i tworzy się kolejny węzeł. Jeśli warunek podziału nie jest spełniony (co oznacza, że został stworzony liść), algorytm wraca do ostatniej niedokończonej gałęzi drzewa lub kończy procedurę budowy drzewa.

W poniższych rozważaniach R_m będzie obszarem przestrzeni $\mathcal{S}$ zawierającym obserwacje, które znalazły się w ustalonym węźle m.

Dla problemu klasyfikacji kluczową rolę przy podziale węzła odgrywa częstość występowania danej klasy w węźle, którą wyraża się wzorem:

$$p_i = \frac{\sum_{\{k:x_k \in R_m\}} \mathbb{1}(y_k = i)}{\sum_{k=1}^{n} \mathbb{1}(x_k \in R_m)}.$$

Najczęściej stosowane miary zróżnicowania klas dla ustalonego węzła to:

- **Indeks Giniego**, który wyraża się wzorem:

$$I_G(p) = \sum_{i \in \mathcal{G}} p_i(1 - p_i),$$

  gdzie przez p_i oznaczamy częstość występowania klasy i w węźle. Indeks Giniego można zinterpretować jako prawdopodobieństwo, że losowa obserwacja zostanie zaklasyfikowana błędnie, w przypadku gdy zachowujemy częstość występowania klas w węźle.

- **Entropia**, która jest wyrażona wzorem:

$$E(p) = \sum_{i \in \mathcal{G}} -p_i \log 2p_i.$$

Jeśli dany wierzchołek drzewa jest liściem, to wynik klasyfikacji dla obserwacji, które znalazły się w tym liściu, to najczęściej występująca w nim klasa.

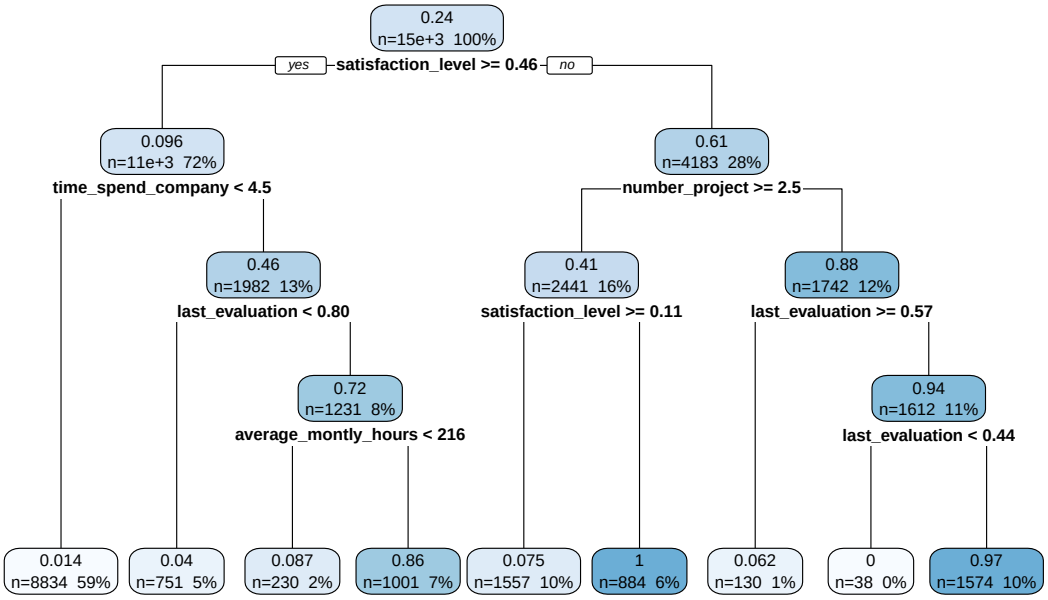
Dla problemu regresji najczęściej wybieraną miarą zróżnicowania węzła jest:

$$Q_m = \frac{\sum_{i:x_i \in R_m} (y_i - \hat{c}_m)^2}{\sum_{k=1}^{n} \mathbb{1}(x_k \in R_m)},$$

gdzie $\hat{c}_m$ to estymator $\hat{y}_i$ dla węzła m, który jest średnią wartością tej zmiennej występującą w węźle:

$$\hat{c}_m = \frac{\sum_{i:x_i \in R_m} y_i}{\sum_{k=1}^{n} \mathbb{1}(x_k \in R_m)}.$$

Jest to również wartość zwracana jako predykcja w liściach.



Rysunek 1.1: Przykładowe drzewo decyzyjne dla zadania klasyfikacji na danych dotyczących odejść pracowników z firmy

Niech Q_m będzie dowolną zdefiniowaną miarą zróżnicowania węzła m. Aby znaleźć najlepszy możliwy podział węzła m należy znaleźć taką cechę X_j i taką wartość s tej zmiennej, która maksymalizuje zysk z podziału węzła:

$$\sum_{k=1}^{n} \mathbb{1}(x_k \in R_m)Q_m - \sum_{k=1}^{n} \mathbb{1}(x_k \in R_L)Q_L - \sum_{k=1}^{n} \mathbb{1}(x_k \in R_L)Q_R,$$

gdzie R_L, to podzbiór zbioru R_m i zawiera obserwacje, które przeszły w węźle m w lewo - spełniły warunek w węźle, natomiast R_L to obserwacje, które przeszły w węźle m w prawo, czyli nie spełniły tej nierówności.

Algorytmy generujące drzewa decyzyjne można znaleźć w środowisku R m. in. w następujących pakietach: `rpart` [43], `tree` [38], `party` [20], `maptree` [45], `partykit` [21], `evtree` [18], `CORElearn` [39], `REEMtree` [40].

Na rysunku 1.1 przedstawiono przykładowe drzewo decyzyjne. Dotyczy ono danych zawierających informacje o pracownikach firmy. Zadanie klasyfikacji polega na oszacowaniu prawdopodobieństwa odejścia pracownika z pracy. Dokładny opis danych znajduje się w rozdziale 4.1.

Rysunek 1.1 pokazuje drzewo klasyfikacyjne stworzone za pomocą funkcji `rpart` z pakietu

Tablica 1.1: Przykładowa obserwacja ze zbioru zawierającego informacje o odejdciach pracowników z firmy. Rzeczywista wartość przewidywanej zmiennej (`left`) wynosi 1.

satisfaction_level	0.64
l last_evaluation	0.91
number_project	4
average_montly_hours	204
time_spend_company	5
Work_accident	0
promotion_last_5years	0
sales	hr
salary	medium
left	1

o tej samej nazwie. Klasyfikowanie nowej obserwacji polega na przydzieleniu jej do odpowiedniego liścia, na podstawie nierówności odpowiednich zmiennych, które znajdują się w węzłach powyżej liścia. W niebieskiej kratce, idąc od góry znajdują się kolejno: bieżąca predykcja, liczba i procent obserwacji przechodzących przez ten węzeł. Poniżej znajduje się nazwa i wartość cechy dla której miara zróżnicowania węzła była najmniejsza oraz dwie gałęzie. Do lewej gałęzi trafiają obserwacje, które spełniają zadaną nierówność w węźle, a do prawej, te które jej nie spełniają.

Rozważmy obserwację o profilu cech określonym w tabeli 1.1 Analizowana obserwacja przy pierwszym podziale przeszła w lewo, tzn. spełnia zadaną w korzeniu nierówność. Początkowa wartość predykcji wynosi 0.24, co wynika z faktu, że w danych jest 24% obserwacji posiadających etykietę o wartości 1. Przejście obserwacji przez pierwszy węzeł zmniejsza predykcję z 0.24 do 0.096. W kolejnym węźle obserwacja przechodzi w prawo, a wartość jej predykcji zwiększy się do 0.46 i w następnym węźle obserwacja przechodzi w lewo, a to ponownie zmniejszy jej predykcję do 0.087, co jest końcową predykcją zwróconą przez ten klasyfikator dla badanej obserwacji.

Zaletą drzewa decyzyjnego jest jego interpretowalność. Analizując węzły można zobaczyć wpływ poszczególnych zmiennych na wynik końcowy dla konkretnej obserwacji a także określić, które zmienne są istotne w modelu oraz na jak wiele obserwacji mają wpływ.

Niewątpliwą wadą drzew jest ich niska dokładność, dlatego pojawiła się potrzeba ulepszenia wyżej przedstawionej metody.

1.2. Komitety klasyfikatorów - bagging i boosting

Drzewo decyzyjne jako samodzielny klasyfikator nie daje najlepszych rezultatów. Jest niestabilne - niewielka zmiana w danych może spowodować istotną zmianę klasyfikatora. Klasyfikatory o niskiej dokładności, niewiele lepsze od losowego przypisania nazywa się *słabymi uczniami*. Bagging i boosting to dwa różne sposoby łączenia wielu słabych klasyfikatorów np. drzew, w taki sposób, aby stworzyć jedną silną regułę decyzyjną.

Bagging to metoda stworzona przez Brainama w 1996 roku. Zgodnie z tą metodą model tworzymy z wielu niezależnych klasyfikatorów, których dokładność jest niewiele lepsza od losowego przypisania klas. Gdy model będzie składał się z odpowiednio dużej liczby klasyfikatorów istnieje duże prawdopodobieństwo, że większość z nich zwróci dobry wynik.[28]

W rzeczywistości, gdy uczymy modele na tych samych danych, korelacja między poszczególnymi drzewami jest duża, co powoduje, że prawdopodobieństwo popełnienia błędu przez

model rośnie. Aby tego uniknąć stosuje się *próby bootstrapowe*. Są to próbki danych o tej samej wielkości co dane wejściowe, przy czym wybierane są one przez losowanie ze zwracaniem obserwacji z wejściowych danych (w pojedynczej próbie niektóre obserwacje mogą się powtarzać). Dzięki temu poszczególne próby różnią się od siebie.

Najbardziej znanym modelem wykorzystującym bagging są lasy losowe. Jest to komitet drzew klasyfikacyjnych. Najlepsze rezultaty dają drzewa bez przycinania, czyli takie, w których liście zwracają ostateczną wartość predykcji np. w przypadku klasyfikacji binarnej w liściach otrzymamy wartość 0 lub 1. Sprawia to, że drzewa te są głębokie. Po wytrenowaniu podanej przez użytkownika liczby drzew, aby zaklasyfikować nową obserwację, model sprawdza wynik zwracany przez każde drzewo. Następnie zwraca wartość, która została wskazana przez większość klasyfikatorów. Optymalne liczby drzew dla zadania klasyfikacji i regresji wynoszą odpowiednio: $\sqrt{n}$ i $n/3$, gdzie n to liczba obserwacji w modelu. Lasy losowe są zaimplementowane między innymi w następujących pakietach: `randomForest, randomForestSRC, ranger, Rborist, h2o, RRF, rotationForest`. `randomForest` [30], `randomForestSRC` [22, 23, 24], `ranger` [46], `Rborist` [41], `h2o` [29], `RRF` [7, 8, 9], `rotationForest` [2].

Bagging daje dużo lepsze rezultaty niż pojedynczy słaby klasyfikator, jednak wybierając próby bootstrapowe model zdaje się na losowość. Problem łączenia słabych klasyfikatorów można rozwiązać w lepszy sposób - wyciągając wnioski z każdego kolejnego klasyfikatora. Obserwacje błędnie zaklasyfikowane w poprzednich uczniach powinny być traktowane jako bardziej istotne w kolejnych iteracjach modelu. Dzięki temu model w każdym kroku staje się coraz dokładniejszy, a poprawa modelu wynika z kontrolowanej zmiany wag poszczególnych obserwacji. W tej sytuacji istnieje jednak niebezpieczeństwo przeuczenia modelu.

Opisana wyżej metoda została nazwana boostingiem od angielskiego słowa *boost-* zwiększać, poprawiać, gdyż w każdej iteracji zwiększa się dokładność modelu. Boosting stał się odpowiedzią na teoretyczne pytanie postawione przez Krearns'a i Valiant'a w 1988 roku, czy słaby uczeń może być *poprawiony* do silnego ucznia, czyli klasyfikatora o dużej dokładności. Problem ten został rozwiązany w 1990 przez Roberta Schapire'a, jednak dopiero w 1995 wraz z Yoav'em Freudem zaproponowali w pełni użyteczny algorytm AdaBoost, który jest pierwszą, ale nadal jedną z najpopularniejszych implementacji boostingu.[19]

Celem boostingu jest znalezienie komitetu $f(x)$ składającego się z K słabych modeli:

$$f(\mathbf{x}) = \sum_{k=0}^{K} f_k(\mathbf{x}) = \theta_0 + \sum_{k=0}^{K} \theta_k \phi_k(\mathbf{x}),$$

gdzie ϕ_k to pojedynczy model, a θ_k to waga tego modelu. Nowe modele dodaje się sekwencyjnie, tak aby nowy model ϕ_k zwiększał dokładność komitetu $k-1$ wcześniejszych modeli.Znalezienie najlepszego modelu w kroku k sprowadza się do rozwiązania następującego zadania minimalizacji:

$$\{\hat{\theta}_k, \hat{\phi}_k\} = \underset{\{\theta_k, \phi_k\}}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \hat{f}^{(k-1)}(\mathbf{x}_i) + \theta_k \phi_k(\mathbf{x}_i)), \tag{1.1}$$

gdzie n to liczba obserwacji w zbiorze treningowym, a $\hat{f}^{(k-1)}$ to komitet składający się z $k-1$ dotychczas utworzonych słabych uczniów.

Jeżeli założymy, że $\phi(\mathbf{x}_i)$ jest klasyfikatorem binarnym, zwracającym 1 lub -1, a funkcję straty ustalimy jako funkcję eksponencjalną, problem ten można rozwiązać w sposób analityczny, wyznaczając dokładnie optymalne θ i ϕ. W ten sposób otrzymamy wspomniany powyżej algorytm AdaBoost.

Zadanie minimalizacji 1.1 daje się rozwiązać w sposób analityczny dla nielicznych postaci funkcji straty. Dla funkcji straty, które są bardziej skomplikowane stosuje się przybliżone rozwiązanie równania 1.1. Najbardziej znane tego typu metody to gradient boosting i Newton boosting [19, 31].

Pierwsza z wyżej wymienionych metod do optymalizacji równania 1.1 wykorzystuje *metodę gradientu prostego*. Aby zastosować tę metoda rozważana funkcja straty musi być funkcją ściśle wypukłą i klasy C^2. Metoda ta opiera się na obserwacji, że funkcja celu maleje najszybciej w kierunku przeciwnym do gradientu funkcji (Algorytm 1).

Newton boosting wykorzystuje natomiast *metodę Newtona* dla zadania znajdowania minimum zadanej funkcji celu. Ta metoda natomiast do wyznaczania kierunku poszukiwań minimum funkcji wykorzystuje rozwinięcie Taylora drugiego rzędu dla zadanej funkcji (Algorytm 2).

Przedstawione algorytmy opierają się na pozycji [31].

Algorytm 1 Gradient boosting

Wejście: $\mathcal{S}$- zbiór danych treningowych zawierający n obserwacji,
L - funkcja straty,
ϕ - słaby uczeń,
K- liczba iteracji,
η - współczynnik uczenia się.

1. Ustaw $\hat{f}^{(0)}(\mathbf{x}) = \hat{f}_0(\mathbf{x}) = \theta_0 = \text{argmin}_\theta \sum_{i=1}^{n} L(y_i, \theta)$.
2. Dla każdego k, od $k = 1$ do K, wykonaj:
(a) Dla każdej obserwacji $\mathbf{x}_i$ gdzie $i \in \{1, ..., n\}$ oblicz gradient:

$$\hat{g}_k(\mathbf{x}_i) = \frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\bigg|_{(f(\mathbf{x})=\hat{f}^{(k-1)}(\mathbf{x}))}$$

.
(b) Znajdź $\hat{\phi}_k$ tż:

$$\hat{\phi}_k = \underset{\phi,\beta}{\text{argmin}} \sum_{i=1}^{n} [(-\hat{g}_k(\mathbf{x}_i)) - \beta\phi(\mathbf{x}_i)]^2$$

(c) Znajdź $\hat{\theta}_k$, tż:

$$\hat{\theta}_k = \underset{\theta}{\text{argmin}} \sum_{i=1}^{n} L(y_i, \hat{f}^{(k-1)}(\mathbf{x}_i) + \theta\hat{\phi}_k(\mathbf{x}_i))$$

(d) Stwórz nowy model: $\hat{f}_k(\mathbf{x}) = \eta\hat{\theta}_k\hat{\phi}_k$.

(e) Dodaj nowy model do komitetu. Ustaw: $\hat{f}^{(k)}(\mathbf{x}) = \hat{f}^{(k-1)}(\mathbf{x}) + \hat{f}_k(\mathbf{x})$.

Wyjście: Komitet modeli

Algorytm 2 Newton boosting

Wejście: $\mathcal{S}$- zbiór danych treningowych zawierający n obserwacji,
L - funkcja straty,
ϕ - słaby uczeń,
K- liczba iteracji,
η - współczynnik uczenia się.

1. Ustaw $\hat{f}^{(0)}(\mathbf{x}) = \hat{f}_0(\mathbf{x}) = \theta_0 = \text{argmin}_\theta \sum_{i=1}^{n} L(y_i, \theta)$.
2. Dla każdego k, od $k = 1$ do K, wykonaj:
(a) Dla każdej obserwacji $\mathbf{x}_i$ gdzie $i \in \{1, ..., n\}$ oblicz gradient:

$$\hat{g}_k(\mathbf{x}_i) = \left. \frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right|_{(f(\mathbf{x})=\hat{f}^{(k-1)}(\mathbf{x}))}$$

(b) Dla każdej obserwacji $\mathbf{x}_i$ gdzie $i \in \{1, ..., n\}$ oblicz hesjan:

$$\hat{h}_k(\mathbf{x}_i) = \left. \frac{\partial^2 L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)^2} \right|_{(f(\mathbf{x})=\hat{f}^{(k-1)}(\mathbf{x}))}$$

(c) Znajdź $\hat{\phi}_k$ tż:

$$\hat{\phi}_k = \underset{\phi}{\text{argmin}} = \sum_{i=1}^{n} \frac{1}{2} \hat{h}_k(\mathbf{x}_i)[(-\frac{\hat{g}_k(\mathbf{x}_i)}{\hat{h}_k(\mathbf{x}_i)}) - \phi(\mathbf{x}_i)]^2$$

(d)Stwórz nowy model: $\hat{f}_k(\mathbf{x}) = \eta \hat{\phi}_k$.

(e) Dodaj nowy model do komitetu. Ustaw: $\hat{f}^{(k)}(\mathbf{x}) = \hat{f}^{(k-1)}(\mathbf{x}) + \hat{f}_k(\mathbf{x})$.

Wyjście: Komitet modeli

1.3. XGBoost - eXstreme Gradient Boosting

Twórcami metody XGBoost są Tianqi Chen i Carlos Guestrin. Pierwszy algorytm pojawił się w 2014 roku i obsługiwał klasyfikację binarną i regresję. Projekt ten jest dostępny jako pakiet open source na stronie internetowej [13]. Sama idea rozwiązania Chen'a i Guestrina była znana już wcześniej w literaturze, a w szczególności wywodzi się z pracy J.H.Friedmana *Greedy Function Approximation: A Gradient Boosting Machine*. Znaczącą innowacją wprowadzoną przez autorów XGBoost, w porównaniu do poprzednich znanych implementacji gradient boostingu, było wprowadzenie czynnika regularyzacji funkcji celu, czyli kary za złożoność modelu.

Głównym źródłem informacji w poniższych rozważaniach jest [11].

Niech $\mathbf{S} = \{(\mathbf{x}_i, y_i)\}$ będzie zbiorem danych treningowych składającym się z n obserwacji $\mathbf{x}_i \in \mathbb{R}^p$, gdzie$i \in \{1, \cdots, n\}$, opisanych przez p cech, oraz przypisanych do obserwacji zmiennych objaśnianych $y_i \in \mathbb{R}$. Komitet klasyfikatorów będzie utworzony z drzew typu CART (*Classification and Regression Trees*) opisanego w rozdziale 1.1. *Drzewa regresyjne*, w każdym liściu zwracają ciągłą wartość, która w tym przypadku będzie traktowana jako waga danego drzewa dla danej obserwacji.

Niech $\mathcal{F} = \{f(x) = w_{q(x)}\}$ będzie przestrzenią składającą się z K drzew, gdzie funkcja $q : \mathbb{R}^p \longrightarrow T$ opisuje strukturę każdego drzewa, T to liczba liści w danym drzewie, a w to wagi liści, które są zależne od struktury drzewa. Wartość $f_k(\mathbf{x}_i)$ zwraca wagę liścia, do którego trafiła i-ta obserwacja, w k-tym drzewie. Zatem końcowa predykcja dla i-tej obserwacji będzie określona wzorem:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(\mathbf{x}_i). \tag{1.2}$$

Aby znaleźć zbiór najlepszych drzew decyzyjnych należy zminimalizować regularyzowaną funkcję celu składającą się z funkcji straty i regularyzacji.

$$\mathcal{L}(\hat{y}_i) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_i). \tag{1.3}$$

W powyższym wzorze l jest wypukłą funkcją straty, klasy C^2, a $\Omega(f)$ to regularyzacja opisana następującym wzorem:

$$\Omega(f_i) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2, \tag{1.4}$$

gdzie T to liczba liści, a w_i to wagi liści w i-tym drzewie.

W XGBoost jest możliwość zastosowania regularyzacji postaci:

$$\Omega(f_i) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + \alpha \sum_{j=1}^{T} |w_j|. \tag{1.5}$$

Ta praca ograniczy się jednak do regularyzacji postaci 1.4.

Zastosowanie regularyzacji ma zapobiec przetrenowaniu modelu, poprzez karanie modeli bardziej złożonych, które mają tendencje do przeuczania się. Poprzez odpowiednie ustawienie parametrów w Ω powinniśmy otrzymywać prostsze w strukturze i zarazem dokładniejsze klasyfikatory. Bez zastosowania regularyzacji algorytm XGBoost sprowadza się do tradycyjnego boostingu drzew, wykorzystującego metodę Newtona.

Znalezienie minimum funkcji 1.3 dla wszystkich drzew na raz byłoby trudnym zadaniem, dlatego proces znajdowania najlepszego drzewa jest wykonywany iteracyjnie. W każdym kroku

szukamy takich wag w nowym drzewie, aby funkcja straty była najmniejsza. Wartość zwracana przez model zmienia się w każdym kroku zgodnie z następującą regułą:

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0, \\
\hat{y}_i^{(1)} &= f_1(\mathbf{x}_i) = \hat{y}_i^{(0)} + f_1(\mathbf{x}_i), \\
\hat{y}_i^{(2)} &= f_1(\mathbf{x}_i) + f_2(\mathbf{x}_i) = \hat{y}_i^{(1)} + f_2(\mathbf{x}_i), \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(\mathbf{x}_i) = \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i).
\end{aligned} \tag{1.6}$$

Korzystając z powyższego algorytmu można zapisać funkcję celu dla t-tego drzewa, przy $t-1$ ustalonych drzewach jako:

$$\begin{aligned}
\mathcal{L}(\hat{y}_i)^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t) + constant.
\end{aligned} \tag{1.7}$$

Dla kwadratowej lub eksponencjalnej funkcji straty minimalizację równania 1.7 można przeprowadzić wprost, wykonując odpowiednie rachunki. Jednak dla bardziej skomplikowanych funkcji straty jest to niemożliwe. Aby rozwiązać to zadanie można przybliżyć równanie 1.7 korzystając z rozwinięcia funkcji w szereg Taylora:

$$\mathcal{L}(\hat{y}_i)^{(t)} \approx \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t) + constant, \tag{1.8}$$

gdzie

$$\begin{aligned}
g_i &= \left. \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i} \right|_{\hat{y}_i = \hat{y}_i^{(t-1)}}, \\
h_i &= \left. \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i^2} \right|_{\hat{y}_i = \hat{y}_i^{(t-1)}}.
\end{aligned} \tag{1.9}$$

$l(y_i, \hat{y}_i^{(t-1)})$ jest znane, stąd po usunięciu stałych zadanie minimalizacji równania 1.7 sprowadza się do następującej postaci:

$$\sum_{i=1}^{n} [g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t). \tag{1.10}$$

Obserwacje, które trafiły w dodawanym w kroku t drzewie do jednego liścia, otrzymują w tym drzewie taką samą wagę w (ang. *score*), co uwzględniamy w poniższym równaniu:

$$\begin{aligned}\mathcal{L}(\hat{y}_i)^{(t)} &\approx \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i)w_j^2] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T \\ &= \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T,\end{aligned} \tag{1.11}$$

gdzie zbiór $I_j = \{i : q(x_i) = j\}$ to zbiór wszystkich obserwacji, które zostały przypisane do j-tego liścia. Dla uproszczenia zapisu wprowadzono zmienne

$$G_j = \sum_{i\in I_j} g_i, \tag{1.12}$$

$$H_j = \sum_{i\in I_j} h_i. \tag{1.13}$$

Minimalizacja równania 1.11 sprowadza się do minimalizacji funkcji kwadratowych z w_j jako zmienna postaci:

$$z(w_j) = G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2.$$

Dla ustalonego j, funkcja $z(w_j)$ osiąga minimum w punkcie:

$$w_j^* = -\frac{G_j}{H_j + \lambda}. \tag{1.14}$$

Powyższa wartość to najkorzystniejsza możliwa waga w j-tym liściu dodawanym do modelu drzewa. Wartość funkcji celu dla optymalnych w^* wynosi:

$$\mathcal{L}^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T. \tag{1.15}$$

Powyższe równanie może być rozpatrywane jako miara oceniająca jakość drzewa t. Jest również wykorzystywane przy znajdowaniu najlepszego podziału węzła. Rozpatrywanie wszystkich możliwych struktur q drzew byłoby niewykonalne, dlatego w XGBoost wykorzystuje się algorytm zachłanny, który startuje od pojedynczego liścia. W każdym kroku dodaje nowe gałęzie, tak, aby zysk z utworzenia węzła był największy, czyli wybiera taki podział węzła, aby funkcja straty w danej chwili była najmniejsza. Stąd kryterium podziału dla ustalonego węzła przyjmuje postać:

$$Gain = z(w_{L+R}^*) - z(w_L^*) - z(w_R^*) - \gamma = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma. \tag{1.16}$$

W powyższym równaniu L i R oznaczają zbiór obserwacji, które trafiają odpowiednio do lewego i prawego węzła. Algorytm tworzy nowy węzeł, wtedy gdy $Gain > 0$, czyli w szczególności gdy:

$$Gain_{real} = z(w_{L+R}^*) - z(w_L^*) - z(w_R^*) > 0,$$

czyli funkcja celu po utworzeniu nowego węzła jest mniejsza niż przed jego utworzeniem. Parametr γ pozwala tutaj kontrolować, jak duży zysk ma przynieść podział węzła. Tworzenie węzłów dla których $Gain_{real}$, jest mały np. bliski zeru jest nieopłacalne, model przez dodatkowy węzeł jest bardziej złożony, co pociąga za sobą większą złożoność obliczeniową i większe prawdopodobieństwo przeuczenia się modelu, przy czym dokładność modelu zwiększa się nieznacznie. Dlatego przy tworzeniu modelu należy ustawić odpowiedni parametr γ, aby w drzewach brane były pod uwagę tyko węzły dla których $Gain_{real}$ jest odpowiednio wysoki. W dalszej części pracy przez słowo Gain będzie oznaczana wartość równania 1.16.

1.4. XGBoost - implementacja

Model XGBoost w środowisku R można budować z pomocą pakietu `xgboost` [14] przy użyciu funkcji `xgboost` lub `xgb.train`. Wykaz najważniejszych parametrów funkcji znajduje się w tabeli 1.2.

Istotnym problemem, jeśli chodzi o implementację metody XGBoost jest znajdywanie odpowiedniego warunku podziału węzła. Przeszukiwanie wszystkich możliwych podziałów każdej cechy w modelu byłoby bardzo czasochłonne, a w przypadku zmiennych ciągłych prawie niemożliwe, dlatego twórcy tej metody opracowali kilka sposóbów szukania kandydatów na podziały, spośród których jest wybierany ten dla którego *Gain* jest najwyższy.

W pakiecie `xgboost`, jeśli chodzi o szukanie najlepszych kandydatów na warunek podziału węzła, w zależności od sytuacji wykorzystywanych jest kilka rozwiązań: dokładny algorytm zachłanny, przybliżony algorytm wykorzystywany przy dużych danych zawierających zmienne ciągłe, specjalna wersja dla danych rzadkich oraz metoda oparta na histogramie.

Gdy zbiór danych jest mały, co w tym przypadku oznacza, że model można stworzyć przy użyciu jednego urządzenia, XGBoost wykorzystuje dokładny algorytm zachłanny *exact greedy algorithm* (Algorytm 3). Algorytm ten sprawdza wszystkie możliwe wartości dla wszystkich możliwych cech i wybiera taką cechę i taką jej wartość, aby jej *Gain* był najwyższy. Rozwiązanie to jest najskuteczniejsze - zawsze wybiera możliwy najlepszy podział, ale jest to mało efektywny algorytm i wykonanie go dla dużych danych często jest niemożliwe.

Algorytm 4 przedstawia przybliżony algorytm znajdowania podziałów, który zamiast sprawdzać każdy możliwy podział dla każdej zmiennej, wyznacza dla każdej cechy percentyle i wybiera z nich taki, który ma najwyższą wartość *Gain*. Wyznaczanie percentyli może się odbywać *globalnie*, czyli są one ustalone dla całego drzewa na raz, lub *lokalnie*, czyli dla każdego nowego węzła osobno (co jest mniej efektywne). W zależności od tego jaka opcja zostanie wybrana punkt 2 w Agorytmie 4 powinien być wykonywany odpowiednio przed rozpoczęciem budowania nowego drzewa, lub przed tworzeniem nowego węzła.

XGBoost zawiera również osobny algorytm dla danych rzadkich. Takie dane mogą charakteryzować się tym, że mają wiele brakujących wartości lub tym, że dla wielu przypadków obliczane w powyższych algorytmach statystyki - G lub H (opisane równaniami 1.9) mogą być równe zero. Dane rzadkie mogą też być wyprodukowane, w przypadku gdy początkowe dane zawierają wiele cech kategorycznych, które do pracy z XGBoost powinny być zamienione na odpowiednią liczbę zmiennych binarnych. Metoda ta nazywa się z ang.*one-hot-encoding*.

Największym problemem jest zadecydowanie, co robić z brakującymi zmiennymi. Algorytm XGBoost bardzo dobrze radzi sobie z tą kwestią. Uczy się na podstawie danych, w którą gałąź, obserwacje z brakującymi wartościami dla danej cechy powinny przechodzić w każdym węźle. Metoda nie rozważa jako kandydatów na podziały brakujących wartości, dzięki czemu jest mniej złożona obliczeniowo. Algorytm ten może być również wykonywany w wersji przybliżonej wykorzystującej percentyle analogicznie do Algorytmu 4.

Tablica 1.2: Najważniejsze argumenty funkcji `xgboost / xgb.train` z pakietu `xgboost`

`params`	lista parametrów, której najważniejsze elementy są opisane w tabeli 1.3
`data`	zbiór danych treningowych (funkcja `xgb.train` wymaga użycia zbioru danych w postaci `xgb.DMatrix`)
`nrounds`	maksymalna liczba iteracji
`watchlist`	lista zbiorów danych, wykorzystanych do oceny modelu (tylko dla `xgb.train`)
`obj`	funkcja celu, wykorzystana przy tworzeniu modelu, dostosowana przez użytkownika (tylko dla `xgb.train`)
`feval`	funkcja oceny modelu dostosowana przez użytkownika (tylko dla `xgb.train`)
`early_stopping_rounds`	liczba iteracji, po której zatrzymywane jest trenowanie modelu w sytuacji, gdy model nie polepsza już swojej dokładności
`maximize`	`TRUE/FALSE`- opcja pozwalająca na wybór wyników oceny zgodnie z zasadą: im większa metryka tym lepszy model
`label`	zmienna objaśniana (tylko w przypadku, gdy zbiorem danych jest macierz)
`missing`	informacja, w jaki sposób model powinien traktować brakujące wartości
`weight`	wektor wag dla obserwacji

Tablica 1.3: Lista najważniejszych parametrów zmiennej `params` z funkcji `xgboost`

`booster`	model użyty do tworzenia boostingu. Dostępne opcje `gbtree` (domyślnie) i `gblinear`
`eta`	*learning rate* wskaźnik uczenia. (Domyślnie 0.3)
`gamma`	liczba, określająca minimalną wartość funkcji celu, pozwalająca na kolejny podział wierzchołka w drzewie.
`max_depth`	maksymalna głębokość tworzonych drzew (domyślnie 6)
`min_child_weight`	minimalna suma wag obserwacji znajdujących się w węźle, wymagana do jego utworzenia.
`subsample`	współczynnik wielkości próbki obserwacji użytych podczas tworzenia modelu
`colsample_bytree`	współczynnik liczby cech użytych do tworzenia pojedynczego drzewa w modelu
`lambda`	regularyzacja L^2 wag liści w danym drzewie
`alpha`	regularyzacja L^1 wag liści w danym drzewie
`objective`	rodzaj problemu uczenia maszynowego. Dostępne opcje: regresja liniowa, regresja logistyczna, binarna regresja logistyczna z wynikiem jako prawdopodobieństwo, binarna regresja logistyczna z wynikiem jako log-szansa, klasyfikacja wieloklasowa z funkcją softmax jako funkcją celu, ranking, odpowiednio: `reg:linear, reg:logistic,binary:logistic,binary:logitraw, multi:softmax, rank:pairwise`
`base_score`	domyślna początkowa predykcja obserwacji
`eval_metric`	miara zgodnie, z którą będzie oceniany zbiór walidacyjny

Algorytm 3 Dokładny algorytm zachłanny znajdowania kryterium podziału węzła

Wejście: zbiór I_m obserwacje znajdujące się w węźle m, który chcemy podzielić,
p-liczba cech,
l- liczba obserwacji w węźle, który chcemy podzielić

1. Ustaw $Gain \leftarrow 0$, $G_{L+R} \leftarrow \sum_{i \in I_m} g_i$, $H_{L+R} \leftarrow \sum_{i \in I_m} h_i$.
2. Dla każdej cechy k: gdzie $k \in \{1, ..., p\}$ wykonaj:
(a) ustaw $G_L \leftarrow 0$, $H_L \leftarrow 0$,
(b) posortuj obserwacje rosnąco względem cechy k,
(c) dla każdej obserwacji j,gdzie $j \in \{1, ..., l\}$, wykonaj w kolejności ustalonej w (2b):
$G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
$G_R \leftarrow G_{L+R} - G_L$, $H_R \leftarrow H_{L+R} - H_L$
$Gain \leftarrow max(Gain, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda})$

Wyjście: Kryterium podziału węzła z najwyższą wartością $Gain$

Algorytm 4 Przybliżony algorytm znajdowania kryterium podziału węzła

Wejście:zbiór I_m obserwacje znajdujące się w węźle m, który chcemy podzielić,
p-liczba cech,
l- liczba obserwacji w węźle, który chcemy podzielić

1. Dla każdej cechy k, gdzie $k \in \{1, ..., p\}$ wyznacz percentyle dla cechy k.
Niech zbiór $S_k = \{s_{k1}, s_{k2}, ..., s_{k100}\}$ będzie zbiorem percentyli.
2. Dla każdej cechy k, gdzie $k \in \{1, ..., p\}$, wykonaj:
$G_{kv} \leftarrow \sum_{j \in \{j | s_{kv} \geq x_{jk} \geq s_{k(v-1)}\}} g_j$,
$H_{kv} \leftarrow \sum_{j \in \{j | s_{kv} \geq x_{jk} \geq s_{k(v-1)}\}} h_j$,
3. Ustaw $Gain \leftarrow 0$, $G_{L+R} \leftarrow \sum_{i \in I_{L+R}} g_i$, $H_{L+R} = \sum_{i \in I_{L+R}} h_i$.
4. Dla każdej cechy k: gdzie $k \in \{1, ..., p\}$ wykonaj:
(a) ustaw $G_L \leftarrow 0$, $H_L \leftarrow 0$,
(d) dla każdego percentyla v, od $v = 1$, do 100, wykonaj:
$G_L \leftarrow G_L + G_{kv}$, $H_L \leftarrow H_L + H_{kv}$
$G_R \leftarrow G_{L+R} - G_L$, $H_R \leftarrow H_{L+R} - H_L$
$Gain \leftarrow max(Gain, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda})$

Wyjście: Kryterium podziału węzła z najwyższą wartością $Gain$

Algorytm 5 Algorytm znajdowania kryterium podziału węzła dla danych rzadkich

Wejście: I_m - zbiór obserwacji znajdujących się w węźle m, który chcemy podzielić,
$I_k = \{i \in I_m : x_{ik}$ nie jest brakującą wartością$\}$ - zbiór obserwacji, które nie mają brakujących wartości dla ustalonej cechy k,
p - liczba cech.

1. Ustaw $Gain \leftarrow 0$, $G_{L+R} \leftarrow \sum_{i \in I_m} g_i$, $H_{L+R} \leftarrow \sum_{i \in I_m} h_i$.
2. Dla każdej cechy k: gdzie $k \in \{1, ..., p\}$ wykonaj:

Brakujące wartości kierowane są w prawą gałąź
(a) ustaw $G_L \leftarrow 0$, $H_L \leftarrow 0$,
(b) posortuj obserwacje ze zbioru I_k rosnąco względem cechy k,
(c) dla każdej obserwacji j,gdzie $j \in \{1, ..., |I_k|\}$, wykonaj w kolejności ustalonej w 2(b):
$G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
$G_R \leftarrow G_{L+R} - G_L$, $H_R \leftarrow H_{L+R} - H_L$
$Gain \leftarrow max(Gain, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda})$

Brakujące wartości kierowane są w lewą gałąź
(d) ustaw $G_R \leftarrow 0$, $H_R \leftarrow 0$,
(e) posortuj obserwacje ze zbioru I_k malejąco względem cechy k,
(f) dla każdej obserwacji j,gdzie $j \in \{1, ..., |I_k|\}$, wykonaj w kolejności ustalonej w 2(e):
$G_R \leftarrow G_R + g_j$, $H_R \leftarrow H_R + h_j$
$G_L \leftarrow G_{L+R} - G_R$, $H_L \leftarrow H_{L+R} - H_R$
$Gain \leftarrow max(Gain, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda})$

Wyjście: Kryterium podziału z najwyższą wartością $Gain$ oraz domyślny kierunek dla brakujących obserwacji

Kolejnym algorytmem znajdowania kandydatów na warunek podziału węzła jest algorytm oparty na histogramie. Każda ciągła cecha zostaje zdyskretyzowana - podzielona na przedziały równej długości i przy przeszukiwaniu kandydatów na warunek podziału brane są pod uwagę wartości dyskretne tej zmiennej. Użytkownik za pomocą argumentu `max_bin` określa liczbę przedziałów. Im wyższa jest liczba podziałów tym dokładniejszy jest model, ale tym samym bardziej złożony obliczeniowo.

Użytkownik XGBoost może ustawić odpowiadający mu algorytm za pomocą parametru `tree_method`, ustawiając go jako `exact` dla Algorytmu 3, `approx` dla Algorytmu 4,`hist` dla algorytmu opartego na histogramie, a także `gpu_exact` i `gpu_hist` dla wersji powyższych algorytmów wykorzystujących moc obliczeniową kart graficznych.

Oprócz opisanej powyżej regularyzacji w XGBoost zastosowano również inne metody, które mają zapobiegać przeuczaniu się modelu. Jest to z ang. *shrinkage* oraz próbkowanie obserwacji i kolumn. Pierwsza metoda polega na ustaleniu przez użytkownika wskaźnika uczenia się (`learning_rate`), który w dokumentacji XGBoost jest oznaczony jako `eta`. Jest to waga dla każdego drzewa utworzonego w modelu, która niweluje wpływ pojedynczych drzew na cały model. Gdy zmniejsza się wskaźnik uczenia należy zwiększyć proporcjonalnie liczbę drzew w modelu i analogicznie w drugą stronę. Im mniejszy wskaźnik uczenia tym lepiej - model jest bardziej odporny na przeuczenie, ale niestety jest też bardziej złożony obliczeniowo.

Próbkowanie polega natomiast na wybraniu przez model w sposób losowy określonej przez użytkownika liczby zmiennych lub obserwacji i wykorzystaniu ich do tworzenia modelu. Próbkowanie obserwacji odbywa się globalnie - XGBoost wybiera określoną liczbę obserwacji i na ich podstawie buduje cały model. W dokumentacji XGBoost jest to parametr `subsample`. Próbkowanie kolumn może odbywać się przed tworzeniem każdego nowego drzewa lub nowego węzła za co odpowiadają parametry odpowiednio `colsample_bytree` i `colsample_bylevel`. Domyślnie algorytm XGBoost buduje model wykorzystując wszystkie dane.

Opisane powyżej metody, nie są jedynymi użytecznymi algorytmami zaimplementowanymi w bibliotece XGBoost. Możliwości tego pakietu są dużo większe np. istnieje możliwość zmiany modelu branego do komitetu, za pomocą parametru `booster` lub sposobu wzrastania drzewa (parametr `grow_policy`). Biblioteka ta wciąż jest rozwijana i dodawane są do niej nowe funkcjonalności.

1.5. LightGBM

XGBoost przez długi czas wiódł prym wśród stosowanych modeli, ponieważ dawał bardzo dobre wyniki. W 2017 roku pojawiła się jednak alternatywa dla XGBoost'a w postaci pakietu LightGBM. Jest to praca zbiorowa, a jej autorami są: Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma , Qiwei Ye , Tie-Yan Li i jest to projekt dostępny na stronie internetowej [27]. Autorzy zastosowali kilka ulepszeń w porównaniu do ówczesnego XGBoost - przede wszystkim zastosowali algorytmy, które przyśpieszają działanie tej metody. Ten podrozdział opiera się na pozycji [26].

Model lightGBM można zbudować za pomocą funkcji `lightgbm, lgb.train`. Najważniejsze parametry tych funkcji są analogiczne do parametrów przedstawionych w tabelach 1.2, 1.3 dotyczących modelu XGBoost.

Najbardziej obciążającym obliczeniowo zadaniem w dostępnych metodach gradient boostingu drzew jest obliczanie miary zróżnicowania w węźle dla każdej możliwej wartości dla wszystkich dostępnych cech. Mimo zastosowanych w XGBoost'e i w innych pakietach algorytmów zmniejszających liczbę kandydatów na kryterium podziału węzła, złożoność tych rozwiązań nadal nie jest zadowalająca. Twórcy LightGBM wprowadzili dwie nowe metody

Algorytm 6 Gradient-based One-Side-Sampling(GOSS)

Wejście: $\mathcal{S}$- zbiór danych,
K - liczba iteracji,
a - współczynnik wielkości próby dla danych z dużym gradientem,
b - współczynnik wielkości próby dla danych z małym gradientem,
L - funkcja straty,
n - liczba obserwacji.

1. Ustaw wagę dla każdej obserwacji jako 1.
2. Dla każdego k, od $k = 1$ do K:
(a) Oblicz aktualną predycję modelu.
(b) Oblicz wartość funkcji straty dla predykowanych wartości.
(c) Posortuj obserwację malejąco pod względem wartości funkcji straty i stwórz nowy zbiór danych składający się z:
(c1) $top = a \cdot n$ najważniejszych obserwacji,
(c2) próbki o wielkości $b \cdot n$ obserwacji wylosowanych spośród reszty obserwacji.
(d) Dla wszystkich spośród $b \cdot n$ obserwacji, wylosowanych w (c2), ustaw wagę $\frac{1-a}{b}$, dla reszty pozostaw poprzednią wagę.
(e) Stwórz nowy model biorąc pod uwagę funkcję straty poprzedniego klasyfikatora i wagi dla zbioru treningowego.
(f) Dodaj model do komitetu.

Wyjście: Komitet modeli

mające na celu zmniejszenie liczby kandydatów na podziały w każdym węźle: *Gradient-based One-Side Sampling (GOSS)* i *Exclusive Feature Bundling (EFB)*. Pierwsza metoda wyszukuje obserwacje, które mają mały wpływ nadokładność modelu i spośród nich wybiera tylko niewielką próbkę, z której wraz z istotnymi obserwacjami, będzie budowany model. Metoda EFB natomiast redukuje liczbę cech, wyszukując cechy które wzajemnie się wykluczają, czyli przyjmują niezerowe lub brakujące wartości dla różnych obserwacji i łączą te zmienne w odpowiedni sposób.

Algorytm Gradient-based One-Side Sampling (Algorytm 6) wyszukuje obserwacje, które są mniej istotne w budowanym modelu. Zabieg ten jest inspirowany pierwotną ideą boostingu, gdzie w każdym klasyfikatorze obserwacje są ważone. Obserwacje które były błędnie zakwalifikowane w poprzednim klasyfikatorze mają większą wagę, a te dobrze zakwalifikowane - mniejszą, gdyż w kolejnym klasyfikatorze są mniej istotne. W boostingu drzew nie ma wag dla obserwacji, jednak analiza 1.16 pozwala zauważyć, że podobną funkcję do wag pełni iloraz pochodnych obserwacji. Jeżeli wartość ilorazu jest mała to zwiększenie wartości *Gain* jest niewielkie, dlatego można pomijać te zmienne. Całkowite zlikwidowanie tych obserwacji mogłoby źle wpłynąć na dokładność modelu, dlatego losuje się spośród nich próbkę, która jest brana pod uwagę przy tworzeniu modelu. Aby skorzystać z tego algorytmu powinno ustawić się parametr `boosting` jako `goss`.

Wielowymiarowe dane bardzo często są reprezentowane przez macierze rzadkie. Posiadają cechy, które w przypadku wielu obserwacji są zerowe lub nie posiadają wartości. Drugie rozwiązanie zastosowane w LightGBM - algorytm Exclusive Feature Bundling (EFB) - wykorzystuje rzadkość takich cech i redukuje ich liczbę. Cechy te są złączane w tzw. pakiet wzajemnie wykluczających się cech (ang.*exclusive feature bundle*). Algorytm ten można włączyć za pomocą parametru `enable_bundle`.

W rzeczywistych danych rzadko zdarza się, że dwie cechy całkowicie wykluczają się wzajemne. Sytuację, gdy dla jednej obserwacji wystąpiły co najmniej dwie niezerowe wartości w pakiecie wzajemnie wykluczających się cech nazwiemy *konfliktem*.

Algorytm Exclusive Feature Bundling (EFB) (Algorytm 7) sprowadza się do stworzenia grafu z cechami jako wierzchołkami i ważonymi krawędziami łączącymi cechy między którymi wystąpił konflikt. Wagi krawędzi to liczba konfliktów między dwoma cechami. W podanym algorytmie użytkownik może określić jak wiele konfliktów między dwoma cechami może zachodzić, za co odpowiada parametr `max_conflict_rate`, w dalszych rozważaniach oznaczony jako M. Po stworzeniu grafu i ustaleniu wag, sortujemy cechy malejąco pod względem stopni wierzchołków, które je reprezentują. Następnie bierzemy każdą cechę i dodajemy ją do już istniejącego pakietu, jeżeli istnieje pakiet, w którym wszystkie cechy mają z nową cechą konflikt co najwyżej M. W innym razie tworzymy nowy pakiet, do którego przypisujemy badaną cechę. Opisany algorytm ma również szybszą wersję, która nie korzysta już z grafu, a zamiast tego sortuje cechy po liczbie niezerowych wartości. Duża liczba niezerowych wartości zwiększa prawdopodobieństwo wielu konfliktów, dlatego stosowanie ulepszonego w ten sposób algorytmu tylko w nieznaczny sposób pogarsza jego działanie.

Po wyznaczeniu cech, które powinny znaleźć się w jednym pakiecie pojawia się problem w jaki sposób je złączyć, tak aby można było odwrócić operację i ze złączonych pakietów wyodrębnić pierwotne cechy. Sposób łącznia cech w pakiet opiera się na fakcie, ze LightGBM korzysta z algorytmu szukania podziałów opartego na histogramie. Dla jednej cechy zarezerwowany jest jeden konkretny przedział wartości pakietu i inne cechy nie posiadają wartości z tego przedziału. Uzyskuje się to przez dodawanie odpowiednich stałych do cech, z których składa się pakiet, w taki sposób aby przedziały wartości tych cech były rozłączne (Algorytm 8).

Algorytm 7 Szukanie pakietów wzajemnie wykluczających się cech - Exclisuve Feature Bundling

Wejście: p - liczba cech,
M - maksymalna liczba konfliktów.

I sposób:
1. (a) Skonstruuj graf $G = (V, E)$, gdzie V to wierzchołki reprezentujące cechy, a E - to krawędzie łączące cechy, które są w konflikcie.
(b) Posortuj cechy malejąco pod względem stopni wierzchołków.

II sposób:
1. Posortuj cechy malejąco pod względem liczby niezerowych wartości.

2. Dla każdej cechy spośród p cech, posortowanych jak w punkcie 1., wykonaj:
(a) Dla każdego istniejącego pakietu:
(a1) sprawdź, czy maksymalna suma konfliktów przekracza liczbę M,
(a2) jeżeli tak, dodaj cechę do pakietu.
(b) Jeżeli nie ma pakietu spełniającego to kryterium, stwórz nowy pakiet.

Wyjście: Zbiór pakietów wzajemnie wykluczających się cech

Algorytm 8 Łączenie wzajemnie wykluczających się cech - Exclisuve Feature Bundling

Wejście: n-liczba obserwacji,
F - pakiet składający się z l wzajemnie wykluczających się cech.

1. Dla każdej cechy f w pakiecie F:
(a) Wyznacz maksymalną wartość cechy.
(b) Zapamiętaj w tabeli $tabelaZakresow$ maksymalną wartość powiększoną o sumę maksymalnych wartości wszystkich poprzednich cech z pakietu.
2. Stwórz nowa cechę $zlaczonyF$.
3. Dla każdej obserwacji j gdzie $j \in \{1, ..., n\}$:
(a) Ustaw $zlaczonyF[j] \leftarrow 0$.
(b) Dla każdej cechy f, spośród $f \in \{1, ..., |F|\}$:
(b1) jeżeli dla obserwacji j, cecha f ma niezerową wartość, dodaj do wartości $zlaczonyF$ dla obserwacji j, wartość cechy f powiększoną o zakres z $tabelaZakresow$ dla cechy $j - 1$.

Wyjście: $zlaczonyF$, $tabelaZakresow$

Rozdział 2

Analiza struktury modelu XGBoost

Modele stosowane w uczeniu maszynowym wraz z rozwojem tej dziedziny stają się coraz bardziej skomplikowane i złożone. Bardzo często użytkownik przyjmuje wyniki bez możliwości oceny, czy model jest poprawny. W prostych modelach np. regresji liniowej badacz sam potrafi zadecydować, czy model działa zgodnie ze zdobytą przez niego wiedzą i ewentualnie odrzucić go, jeśli tak nie jest. W tzw. modelach *black box* (czarnych skrzynkach) tj. lasy losowe, czy XGBoost, użytkownik nie ma możliwości bezpośredniego zinterpretowania modelu, gdyż jest on zbyt skomplikowany. Pomimo tego, modele te z powodu swojej dokładności cieszą się dużym zainteresowaniem. Istnieje jednak zapotrzebowanie na narzędzia, ułatwiające interpretację wyuczonego modelu.

W świecie uczenia maszynowego pojawiły się już biblioteki interpretujące modele np. `randomForestExplainer` [33, 34], `DALEX` [3, 5], `xgboostExplainer` [15, 16]. Pakiet `randomForestExplainer` to narzędzie do badania ważności zmiennych, a także badania interakcji zachodzących w lasach losowych. Biblioteka `DALEX` to zestaw narzędzi pomagających w interpretacji wielu rodzajów modeli. Za pomocą ogólnych metod możemy zbadać ważność zmiennych, warunkową odpowiedź modelu w zależności od ustalonej zmiennej, a także wyjaśnić predykcję dla danej obserwacji. Natomiast pakiety `xgboostExplainer` oraz `lightgbmExplainer` [10] zajmują się wyjaśnianiem wpływu poszczególnych zmiennych na predykcję pojedynczej obserwacji. Innymi pakietami zajmującymi się podobnymi problemami są np.:`ALEPlot` [1], `breakDown` [42, 6], `pdp` [17], `condvis` [32], `lime` [35, 36], `anchors` [37, 44].

W ramach tej pracy magisterskiej został stworzony pakiet `EIX`. Jest to narzędzie do interpretowania modeli XGBoost [11] i LightGBM [26] ze szczególnym uwzględnianiem interakcji między zmiennymi zachodzącymi w modelu. Biblioteka `EIX` pozwala znaleźć interakcje w modelu, zbadać ważność zmiennych i interakcji oraz prześledzić wpływ poszczególnych zmiennych objaśniających na końcową predykcję pojedynczej obserwacji.

Główną zaletą pakietu `EIX`, która odróżnia go on innych rozwiązań dostępnych w środowisku R, jest to, że w analizie modelu uwzględnia interakcje. Pakiety `xgboostExplainer` oraz `lightgbmExplainer` pozwalają jedynie na analizę predykcji pojedynczej obserwacji bez uwzględniania interakcji. Pakiety `xgboost` [14] i `lightgbm` [27] posiadają funkcje badające ważność zmiennych, jednak nie dają one możliwości badania ważności interakcji w modelu. W pakiecie `EIX` do miar już istniejących wcześniej, dodane zostały nowe, pozwalające dokładniej poznać budowę modelu.

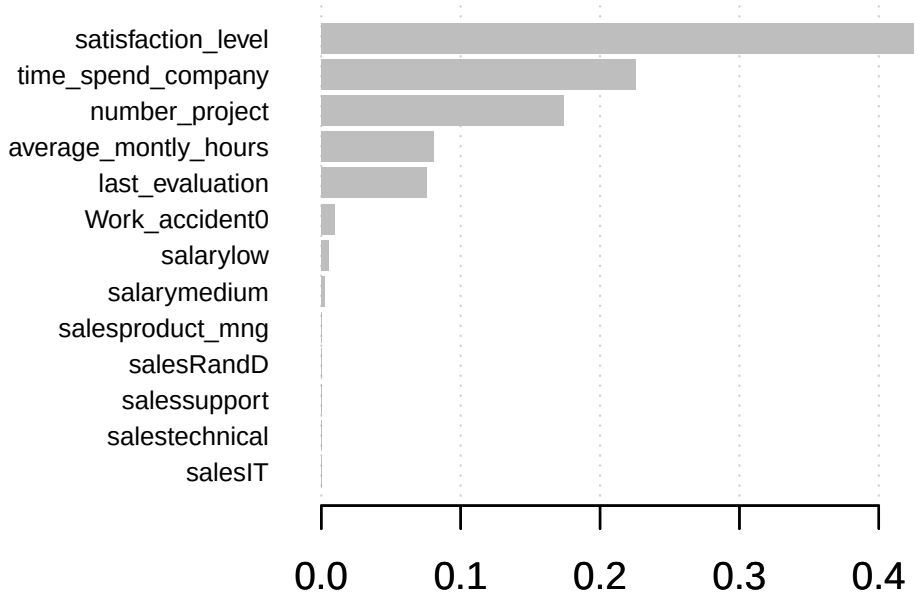
2.1. Ważność zmiennych

Przy ocenie modelu bardzo ważną kwestią jest to, jakie zmienne mają istotny wpływ na końcowe wyniki. Wiedza o tym pozwala ulepszyć model, np przez wyeliminowanie zmiennych, które zgodnie z wiedzą i doświadczeniem badacza nie powinny mieć wpływu na końcowy wynik lub mają w nim zbyt dużą wagę.

W pakiecie `xgboost` [14] jest dostępna funkcja `xgb.importance` która zwraca tabelę z ważnością zmiennych w modelu. Tabela ta zawiera następujące miary zmiennych:

- **Gain** jest to miara określająca dla danej cechy sumę zysków z podziału węzła wyrażonych wzorem 1.16 z wszystkich wystąpień tej cechy w modelu. Miara ta jest normowana przez sumę zysków wszystkich występujących cech w modelu.
- **Cover** jest to suma pochodnych drugiego rzędu obserwacji zaklasyfikowanych do danego węzła (równanie 1.13). Dla kwadratowej funkcji straty będzie to równoważne z liczbą obserwacji przechodzących przez dany węzeł. Metryka ta maleje wraz z rosnącą głębokością węzłów. Jest ona unormowana analogicznie do miary Gain.
- **Frequency** jest to procent, wyrażający liczbę wystąpień danej cechy w całym modelu. Warto zauważyć, ze niektóre cechy mogą występować rzadko, a jednocześnie mieć istotny wpływ na model, co jest spowodowane zastosowaniem wag. Jednocześnie zmienne występujące wiele razy w modelu mogą mieć mały wpływ na końcowy wynik. Dlatego nie należy traktować częstotliwości wystąpień danej cechy bezpośrednio jako miary ważności tej zmiennej.

W pakiecie `xgboost` jest również możliwe zobrazowanie wybranej miary za pomocą wykresu słupkowego przy użyciu funkcji `xgb.plot.importance` (Rysunek 2.1).



Rysunek 2.1: Wykres ważności zmiennych wygenerowany za pomocą funkcji `xgb.plot.importance` pochodzącej z pakietu `xgboost`, dla zbioru danych dotyczących odejścia pracowników z firmy. Na wykresie przedstawiona jest miara Gain.

W bibliotece `EIX` tabelę ważności zmiennych wywołuje się za pomocą funkcji `importanceTable`. Zawiera ona następujące miary ważności zmiennych:

- **sumGain** jest to suma wszystkich wartości Gain występujących w węzłach, w których obserwacje są rozdzielane przez daną zmienną,
- **sumCover** analogicznie do sumGain z wartością Cover,
- **meanGain** jest to sumGain podzielony przez liczbę wszystkich wystąpień cechy w modelu,
- **meanCover** analogicznie do miary meanGain z wartością Cover,
- **meanDepth** jest to średnia ważona głębokość węzła danej cechy, gdzie wagą jest Gain węzła,
- **mean5Gain** jest to średnia z pięciu najwyższych pod względem wartości Gain wystąpień danej cechy w węźle dla danej cechy,
- **countRoots** jest to liczba wystąpień danej cechy w korzeniu,
- **weightedRoot** jest to liczba wystąpień danej zmiennej w korzeniu ważona przez odpowiednie wartości Gain.

2.2. Interakcje

Interakcję zmiennych można zdefiniować jako nieaddytywny wpływ co najmniej dwóch zmiennych na zmienną objaśnianą.

Świadomość tego jakie interakcje zachodzą w danych jest bardzo istotna podczas tworzenia modelu. Jeszcze przed zbudowaniem modelu możemy przekształcić dane uwzględniając znalezione interakcje.

Innym problemem jest znalezienie interakcji w modelu. Dzięki temu zabiegowi, po uwzględnieniu dostępnej wiedzy, można zweryfikować poprawność modelu lub wykorzystać zdobyte informacje przy dalszej pracy z modelem. Po poznaniu interakcji zachodzących w modelu, w pierwszym etapie pracy z danymi, użytkownik może przekształcić dane i stworzyć nowy, dokładniejszy model.

W ramach tej pracy magisterskiej, w celu znalezienia reguły zgodnie, z którą interakcje są uwzględniane w modelach XGBoost zostały przeprowadzone symulacje. Zmienna objaśniana była tworzona zgodnie ze wzorem regresji liniowej i regresji logistycznej, następnie był tworzony model XGBoost i weryfikowane były wyniki.

Dane do symulacji składały się z 2000 obserwacji i z 20 zmiennych z rozkładu Bernoulliego z parametrami $p = 0.5$ i $n = 1$. Liniowy model regresyjny z interakcją zmiennych X_1 i X_2 ma następującą postać:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{1,2}\left(X_1 - \frac{1}{2}\right) \cdot \left(X_2 - \frac{1}{2}\right) + \varepsilon, \tag{2.1}$$

gdzie Y jest zmienną objaśnianą, X_1, X_2 są zmiennymi objaśniającymi, a zmienna ε jest zakłóceniem losowym o rozkładzie normalnym.

Dla zadania klasyfikacji binarnej zastosowano następującą modyfikację:

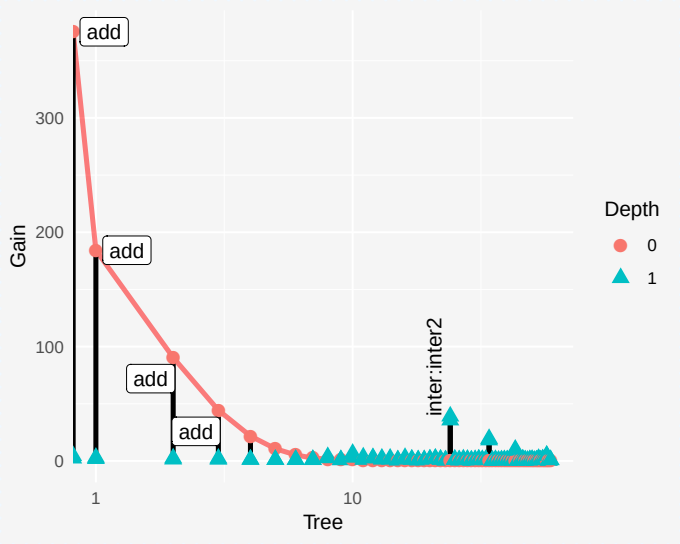
$$p_i = \frac{exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{1,2}\big(X_1 - \frac{1}{2}\big) \cdot \big(X_2 - \frac{1}{2}\big))}{1 + exp(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{1,2}\big(X_1 - \frac{1}{2}\big) \cdot \big(X_2 - \frac{1}{2}\big))}. \tag{2.2}$$

Aby uzyskać efekt losowości dla każdej wartości y_i, odpowiadającej i-tej obserwacji, ostateczną etykietę ustalano zgodnie z rozkładem Bernouliego dla liczby sukcesów $n = 1$ i prawdopodobieństwa sukcesu p_i.

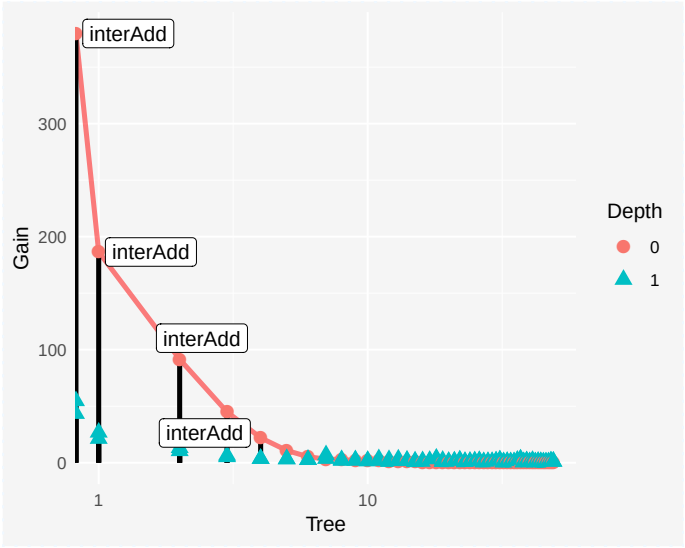
Powyższe wzory zostały wykorzystane do tworzenia zmiennej objaśnianej dla modelu XGBoost. W symulacjach były uwzględnione różne liczby i różne kombinacje zmiennych addytywnych i zmiennych będących w interakcji. Przeprowadzono je zarówno dla regresji jak i klasyfikacji dla różnych parametrów funkcji `xgboost`. Można z nich wyciągnąć następujące wnioski:

- dwie zmienne X_1, X_2 są uznane w modelu za interakcję, gdy jedna zmienna jest dzieckiem drugiej i Gain zmiennej będącej dzieckiem jest większy niż Gain rodzica (Rysunek 2.2 - 2.4).
- interakcje zmiennych, które nie mają jednocześnie wpływu addytywnego na zmienną objaśnianą, ujawniają się w drzewach o wyższym numerze. W tej sytuacji jako samodzielne zmienne mają mały wpływ na dokładność modelu, dlatego najpierw do modelu brane są zmienne mające największy wpływ zgodnie z miarą Gain (Rysunek 2.2).
- w przypadku, gdy dana zmienna objaśniająca X_i ma wpływ addytywny i jednocześnie jest w interakcji z inną zmienną X_j, powyższym sposobem znajdowane są silne interakcje, których wkład w tworzenie modelu jest porównywalny z wkładem addytywnym X_i (Rysunek 2.3,2.4).

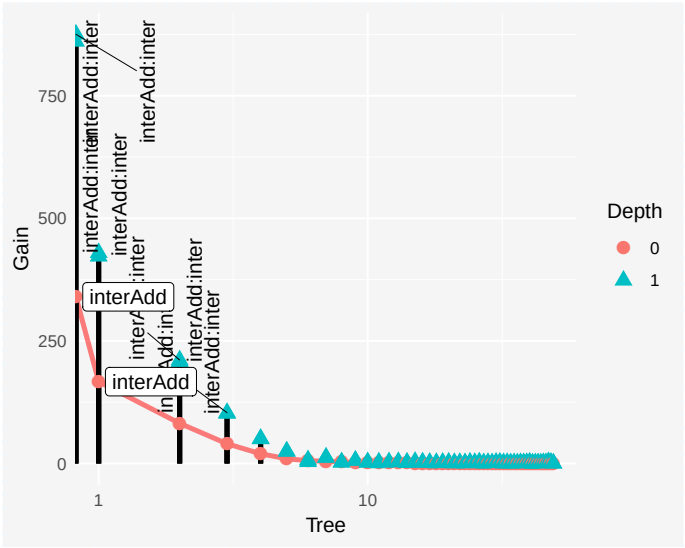
Na rysunkach 2.2- 2.4 są przedstawione przykładowe wyniki symulacji obrazujące powyższą regułę. Sposób interpretacji powyższych wykresów został opisany w podrozdziale 3.1. Zmienne będące w interakcji oznaczono `inter` i `inter2`, zmienną addytywną jako `add`, a zmienną która jednocześnie jest w interakcji z inną zmienną i mającą wpływ addytywny jako `interAdd`. Wykresy te zostały stworzone z pomocą pakietu `EIX`, powstałego w ramach tej pracy magisterskiej. Na osi x znajdują się numery drzew, a na osi y wartość Gain w poszczególnych



Rysunek 2.2: Symulacja modelu XGBoost dla regresji liniowej, gdzie zmienna objaśniana była modelowana przez równanie: $Y = X_1 + (X_2 - \frac{1}{2}) \cdot (X_3 - \frac{1}{2}) + \varepsilon$. W początkowych drzewach ujawnia się duży wpływ zmiennej addytywnej X_1 (na rysunku `add`), dla drzew o wyższych numerach widoczna interakcja zmiennych X_2, X_3 (`inter` i `inter2`).



Rysunek 2.3: Symulacja modelu XGBoost dla regresji liniowej, gdzie zmienna objaśniana była modelowana przez równanie: $Y = X_1 + (X_1 - \frac{1}{2}) \cdot (X_2 - \frac{1}{2}) + \varepsilon$. W drzewie widać duży wpływ zmiennej X_1 (na rysunku `interAdd`), jednak z powodu zbyt małego wpływu interakcji na zmienną objaśnianą, interakcja nie jest wyraźnie widoczna na wykresie.



Rysunek 2.4: Symulacja modelu XGBoost dla regresji liniowej, gdzie zmienna objaśniana była modelowana przez równanie: $Y = X_1 + 4 \cdot (X_1 - \frac{1}{2}) \cdot (X_2 - \frac{1}{2}) + \varepsilon$. W drzewie jest widoczny zarówno duży wpływ zmiennej X_1 (na rysunku `interAdd`), jak i wpływ interakcji, który w tym przypadku ujawnia się w początkowych drzewach.

węzłach drzewa. Każda pionowa linia na wykresie oznacza jedno drzewo. Za pomocą punktów zostały oznaczone węzły, zaś ich kształty informują o głębokości danego węzła w drzewie. W prezentowanych wykresach są przedstawione modele drzew o głębokości 2, różowe kółko oznacza korzeń drzewa, a niebieski trójkąt węzły znajdujące się pod korzeniem. Z interakcją mamy do czynienia w momencie gdy trójkąt znajdzie się nad kółkiem, co oznacza, że Gain dziecka jest wyższy niż Gain korzenia.

W symulacjach badano również wszystkie występujące pary rodzic - dziecko. W tej sytuacji, aby znaleźć pary, które mogą być interakcją sumowano Gain dziecka wszystkich wystąpień takiej pary i wybierano pary o najwyższym współczynniku Gain. Rozpatrywanie takich par w przypadkach określonych w punkcie drugim powyższych wniosków pozwala znaleźć słabe interakcje, jednak zdarza się, że zmienne o wysokiej sumie Gain mogą nie być w rzeczywistości w interakcji, gdyż są zmiennymi o wysokim wpływie addytywnym na model.

Dla interakcji w bibliotece `EIX` są dostępne następujące miary ważności zmiennych spośród wymienionych w podrozdziale 2.1: sumGain, sumCover, mean5Gain, meanGain, meanCover, frequency. Jest również możliwość porównywania ważności pojedynczych cech z interakcjami dwóch zmiennych w jednej tabeli.

2.3. Wyjaśnienie predykcji pojedynczej obserwacji

W pojedynczym drzewie zmienna ma wpływ na predykcję danej obserwacji, gdy obserwacja przejdzie przez węzeł, który rozdziela dana zmienna. W modelu XGBoost jest wiele drzew, a do tego liście posiadają wagi. Obserwacja przy każdym węźle otrzymuje odpowiednią wagę wynikającą ze wzoru 1.16. Wagi te należy uwzględnić przy określaniu jaki wpływ na predykcję pojedynczej obserwacji ma dana zmienna.

Po raz pierwszy zobrazowanie wpływu poszczególnych zmiennych na predykcję wybranej obserwacji dla modelu XGBoost zostało zastosowane w pakiecie `xgboostExplainer` [15, 16].Powstała również wersja tego samego wykresu dla modelu LightGBM w pakiecie `ligthgbmExplainer` [10]. Pakiet `EIX` za zgodą autora pakietu `xgboostExplainer` wykorzystuje i modyfikuje kod zawarty w tym pakiecie, dotyczący opisywanej funkcjonalności.

Uzyskanie współczynników, które określają wpływ zmiennych na predykcję wymaga dodatkowych obliczeń, ponieważ wagi poszczególnych zmiennych w węzłach nie są znane. Funkcja `xgb.model.dt.tree` z pakietu `xgboost` dostarcza tabelę (Tabela 2.1), w której dostępne są następujące informacje:

- `Tree`, `Node` - odpowiednio numer drzewa, numer węzła
- `ID` - numer ID wierzchołka,
- `Feature` - nazwa zmiennej,
- `Split` - warunek podziału węzła,
- `Yes` - ID wierzchołka, do którego przechodzi obserwacja, po spełnieniu warunku podziału,
- `No` - ID wierzchołka, do którego przechodzi obserwacja, gdy nie spełnia warunku podziału,
- `Missing` - ID wierzchołka, do którego przechodzi obserwacja, jeżeli ma ona brakującą wartość dla cechy znajdującej się w węźle,

Tablica 2.1: Wynik funkcji `xgb.model.dt.tree` z pakietu `xgboost`, dla modelu XGBoost na danych dotyczących odejść pracowników z firmy.

	Tree	Node	ID	Feature	Split	Yes	No	Missing	Quality	Cover
1	0	0	0-0	satisfaction_level	0.47	0-1	0-2	0-1	3123.25	3749.75
2	0	1	0-1	number_project	2.50	0-3	0-4	0-3	892.95	1045.75
3	0	2	0-2	time_spend_company	4.50	0-5	0-6	0-5	1284.83	2704.00
4	0	3	0-3	Leaf					0.45	435.50
5	0	4	0-4	Leaf					-0.11	610.25
6	0	5	0-5	Leaf					-0.58	2208.50
7	0	6	0-6	Leaf					-0.05	495.50
8	1	0	1-0	satisfaction_level	0.11	1-1	1-2	1-1	1974.95	3548.88
9	1	1	1-1	Leaf					0.63	221.30
10	1	2	1-2	number_project	2.50	1-3	1-4	1-3	1375.74	3327.58
11	1	3	1-3	Leaf					0.13	563.97
12	1	4	1-4	Leaf					-0.38	2763.61
13	2	0	2-0	satisfaction_level	0.47	2-1	2-2	2-1	1096.64	3234.69
14	2	1	2-1	satisfaction_level	0.11	2-3	2-4	2-3	457.32	973.33
15	2	2	2-2	time_spend_company	4.50	2-5	2-6	2-5	920.84	2261.36

- `Quality` - dla węzłów wartość Gain, dla liści waga,
- `Cover` - miara Cover.

Liczba, która jest wagą liścia jest opisana wzorem 1.14. Aby znaleźć wkład poszczególnych zmiennych należy obliczyć wartość daną wzorem 1.14 dla każdego węzła. Wartości w liściach są znane, zatem przy użyciu wagi lewego i prawego dziecka można obliczyć wagę węzła-rodzica. W tym celu korzystamy z prostego faktu, że

$$G_{L+R} = G_L + G_R,$$

gdzie G jest zdefiniowane wzorem 1.12, jako zbiór L oznaczymy zbiór indeksów, które znalazły się po podziale w węźle w lewym liściu, a zbiór R w prawym liściu.

Powyższe równanie rozszerzymy w następujący sposób:

$$H_{L+R} \cdot \frac{G_{L+R}}{H_{L+R}} = H_L \cdot \frac{G_L}{H_L} + H_R \cdot \frac{G_R}{H_R},$$

gdzie H jest zdefiniowane wzorem 1.13.

Stąd otrzymujemy:

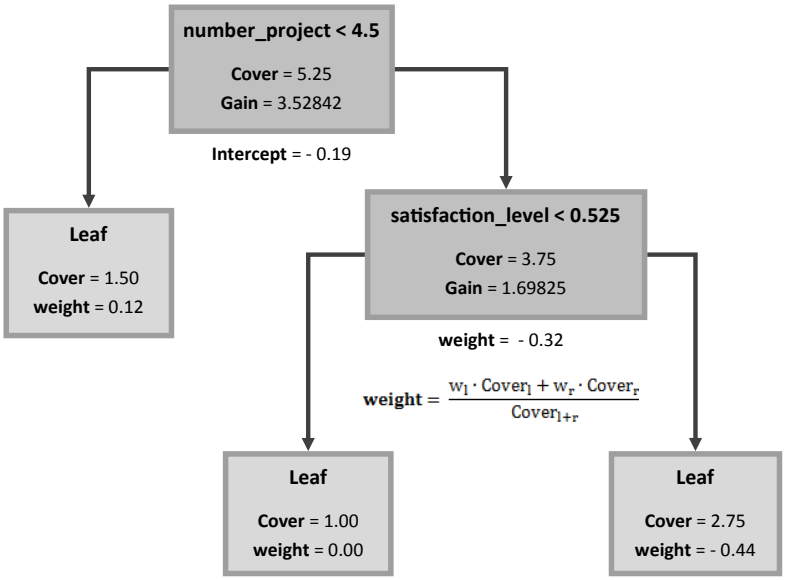
$$H_{L+R} \cdot w_{L+R} = H_L \cdot w_L + H_R \cdot w_R.$$

Dla uproszczenia w rozwiązaniu przyjęto, że $\lambda = 0$.

Statystyka H jest podana w tabelce wygenerowanej przez funkcję `xgb.model.dt.tree` jako wartość Cover, a wagi to dla liści wartość Quality.

Podana powyżej procedura wykonywana jest rekurencyjnie zaczynając od dołu drzewa – od liści dla których znane są wagi.

Tak obliczona waga dla ustalonego węzła jest zależna od zmiennych znajdujących się nad danym węzłem. Dlatego, aby obliczyć samodzielny wpływ zmiennej znajdującej się w węźle należy odjąć od wagi danego węzła wagę jego rodzica. Sytuacja zobrazowana jest schematycznie na rysunku 2.5. Wartość `Intercept` to waga początkowa, która wynika z faktu, że



Rysunek 2.5: Schemat przedstawiający pojedyncze drzewo decyzyjne w modelu XGBoost z uwzględnieniem wag zmiennych znajdujących się w węzłach.

zbiór danych treningowych nie jest zrównoważony - osób które odeszły z firmy jest mniej niż tych które pozostały. Dla obserwacji, która została zakwalifikowana do prawego dolnego liścia, zmienna `number_project` obniżyła predykcję o 0.13, więc wkład tej zmiennej w predykcję wynosi -0.13. Natomiast zmienna `satisfaction_level` obniżyła predykcję o 0.12, i analogicznie wpływ tej zmiennej to -0.12.

Po obliczeniu takich wag we wszystkich drzewach oblicza się ich sumę dla każdej zmiennej i tę wartość uznaje się za wpływ danej zmiennej na predykcję.

Przyjmijmy, że P to zbiór cech opisujący każdą obserwację w danych, użytych do wytrenowania modelu, zbiór J to zbiór wszystkich węzłów w modelu, a J_k będzie zbiorem węzłów, których kryterium spełnia obserwacja k. Niech w_{pj} będzie wagą dla cechy p w węźle o numerze j.

Wpływ zmiennej p na predykcję obserwacji k można opisać następująco:

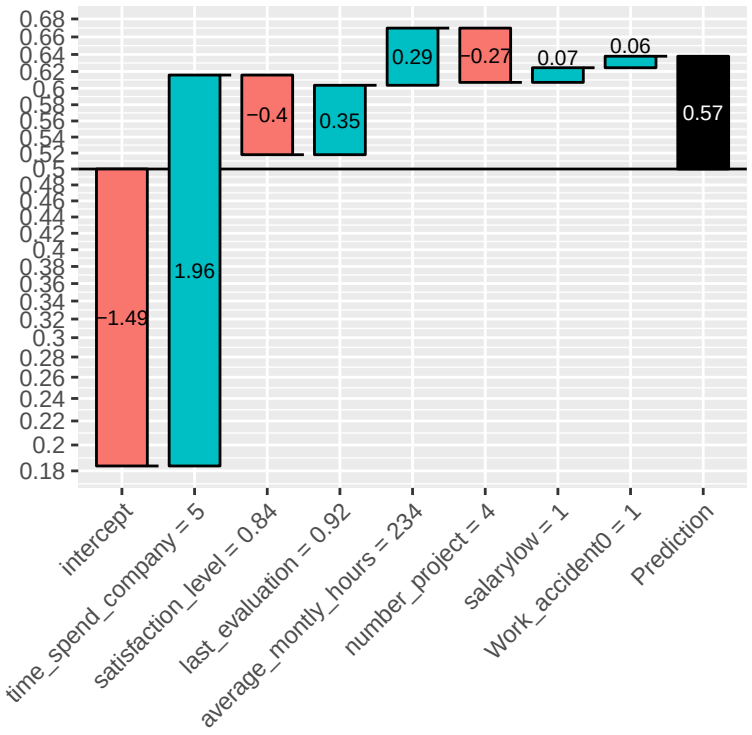
$$w_p(k) = \sum_{j \in J_k} w_{pj}.$$

Wartość predykcji dla obserwacji k ma wtedy następująca postać:

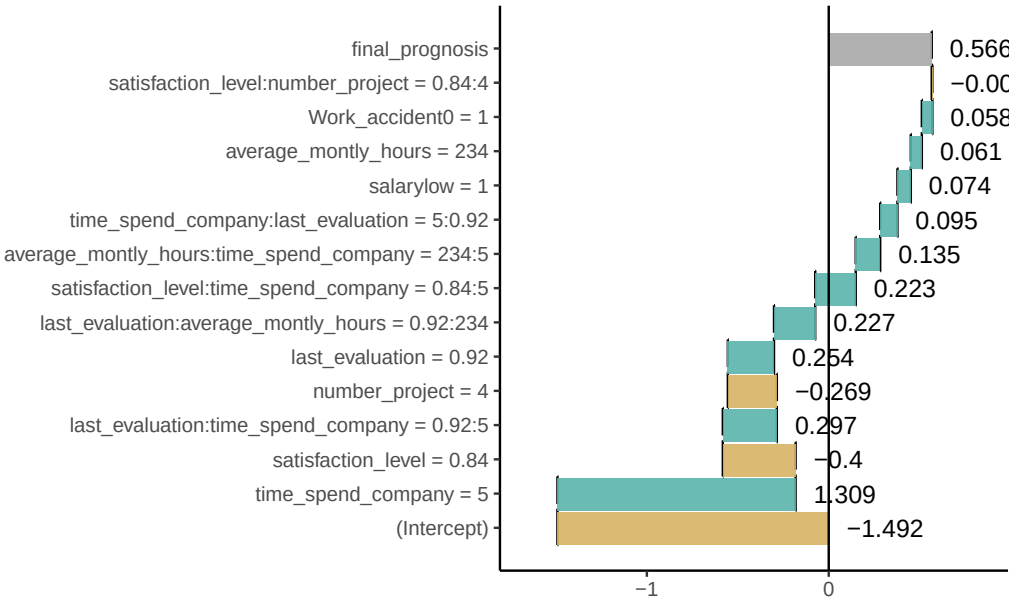
$$w(k) = \sum_{p \in P} w_p(k).$$

Analiza predykcji pojedynczej obserwacji może być zobrazowana na wykresie typu kaskadowego (Rysunek 2.6). Dokładny opis wykresu znajduje się w podrozdziale 3.4.

W pakiecie `EIX` istnieje możliwość uzyskania takiego samego wykresu z uwzględnieniem interakcji, które są znajdowane zgodnie ze schematem opisanym w rozdziale 2.2 (Rysunek 2.7). W tej sytuacji za wagę interakcji uznaje się wagę zmiennej znajdującej się w drzewie niżej, a waga jej rodzica jest uznawana jako waga samodzielnej zmiennej.



Rysunek 2.6: Wyjaśnienie predykcji wybranej obserwacji ze zbioru danych o odejściach pracowników z firmy. Wykres wykonany za pomocą funkcji `showWaterfall` z pakietu `xgboostExplainer`.



Rysunek 2.7: Wyjaśnienie predykcji wybranej obserwacji ze zbioru danych o odejściach pracowników z firmy z uwzględnieniem interakcji zachodzących w modelu. Wykres został wykonany przy użyciu pakietu `EIX` za pomocą funkcji `plot` z wynikiem funkcji `waterfall` jako argumentem

Rozdział 3

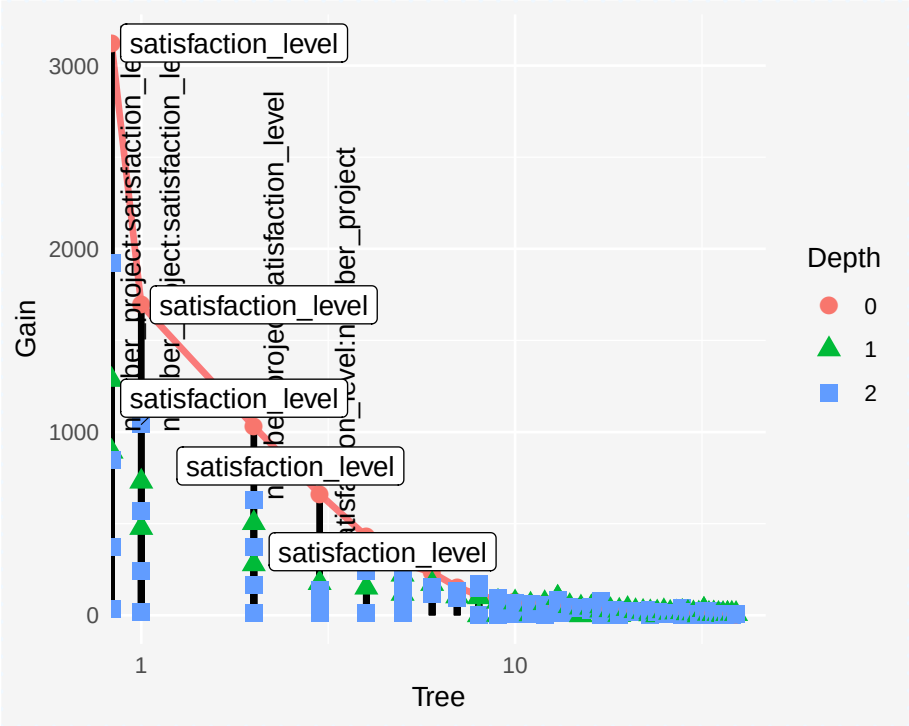
Opis pakietu EIX

Pakiet `EIX` *(Explain Interactions in XGBoost)*, stworzony w ramach tej pracy magisterskiej, umożliwia analizę struktury modelu z uwzględnieniem interakcji. Pakiet ten jest przeznaczony dla modeli XGBoost [11] i LightGBM [26] utworzonych za pomocą pakietów: `xgboost` [14] i `lightgbm` [27]. Działa on zarówno dla regresji liniowej, jak i dla klasyfikacji. Jako dwa niezbędne argumenty należy podać wytrenowany model za pomocą funkcji `xgboost`, `xgb.train`, `lightgbm` lub `lgb.train` oraz zbiór danych wykorzystanych do wytrenowania modelu w postaci `data.table`. Warto zauważyć, że zbiór danych przed trenowaniem należy odpowiednio przygotować: powinien on być w postaci macierzy rzadkiej, a zmienne kategoryczne zakodowane jako zmienne typu `factor`.

3.1. Wizualizacja modelu

W pakiecie `EIX` dostępny jest wykres *lollipop*, który pozwala zwizualizować model w taki sposób, aby ważne zmienne i interakcje były widoczne. Aby go stworzyć należy najpierw wygenerować obiekt *lollipop* przy użyciu funkcji `lollipop`, a następnie wywołać funkcję `plot`, której argumentem będzie stworzony obiekt. Jako dwa niezbędne argumenty funkcji `lollipop` należy podać model XGBoost, który chcemy wyjaśniać i zbiór danych, na których uczony był model w postaci `data.table`. Obiekt *lollipop* składa się z dwóch tabel zawierających najważniejsze informacje o węzłach i korzeniach w drzewach, niezbędnych do stworzenia opisywanego wykresu. Są to między innymi: numer drzewa, wartość Gain, informacje, czy dany węzeł jest częścią interakcji.

Rysunek 3.1 przedstawia wykres *lollipop* dla danych opisujących odejście pracowników z firmy. Na osi x znajdują się numery drzew, natomiast na osi y wartość Gain dla każdego węzła. Każda pionowa kreska oznacza jedno drzewo, na której zaznaczone są wszystkie węzły znajdujące się w drzewie, a kształt punktu symbolizującego węzeł oznacza jego głębokość. W przypadku tego modelu mamy do czynienia z drzewami o głębokości 3. Węzły, które są korzeniami są połączone łamaną linią, co pozwala szybciej je zidentyfikować. Zielony trójkąt oznacza węzły o głębokości 1, natomiast niebieski kwadrat oznacza węzły o głębokości 2. W przypadku przedstawionym na rysunku 3.1, mamy do czynienia z interakcją, jeśli punkt oznaczający węzeł o głębokości 1 (zielony trójkąt) znajduje się nad czerwoną linią (nad korzeniem) lub jeżeli punkt oznaczający węzeł o głębokości 2 (niebieski kwadrat) znajduje się nad zielonym trójkątem. Na wykresie są umieszczone etykiety najważniejszych zmiennych. Etykieta pozioma - zmienne znajdujące się w korzeniach, których wartość Gain jest większa niż 10% największej wartości Gain osiągniętej w modelu. Etykieta pionowa - interakcje, których wartość Gain spełnia analogiczny warunek. Za pomocą parametru `labels` można



Rysunek 3.1: Wizualizacja modelu wykonana za pomocą funkcji `plot` z obiektem *lollipop*, jako argumentem i domyślnymi parametrami dla danych dotyczących odejścia pracowników z firmy. Na wykresie widoczne są najważniejsze zmienne oraz interakcje mające największy wpływ na wyniki modelu.

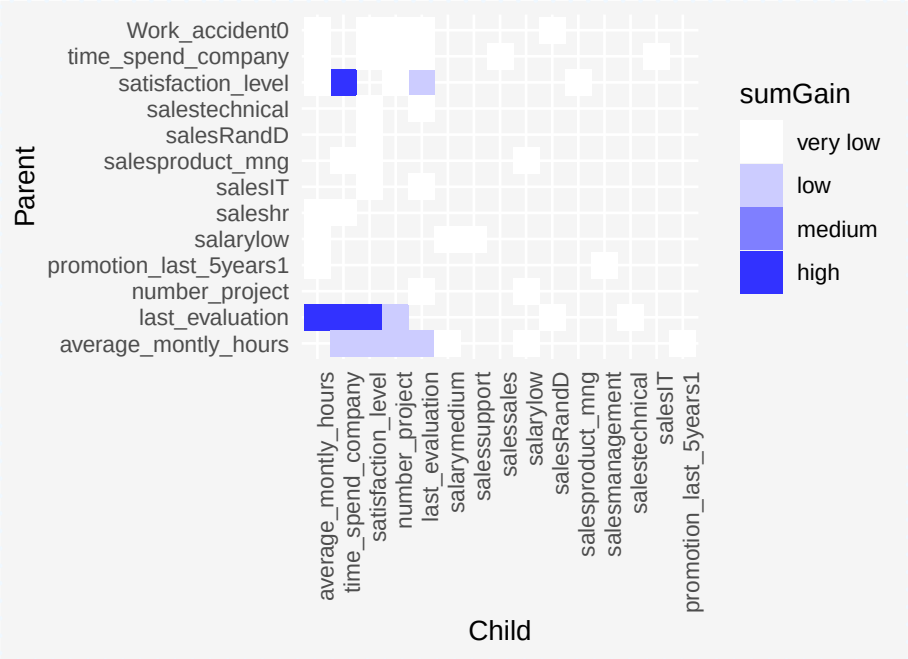
wybrać jakie etykiety mają być wyświetlane. Wartość `"topAll"` (domyślna) tego parametru wyświetla etykiety dla najważniejszych pod względem wartości Gain węzłów w korzeniach jak i w interakcjach, `labels = "interactions"` wyświetla etykiety dla wszystkich interakcji, a `labels = "roots"` dla korzeni.

Wykres przedstawiony na rysunku 3.1 dla większej czytelności jest rysowany w skali logarytmicznej, jest możliwość zastosowania standardowej skali liniowej przez ustawienie parametru `log = FALSE`.

3.2. Interakcje

W celu badania wszystkich interakcji w modelu została stworzona funkcja `interactions`. Aby uruchomić powyższą funkcję należy podać model XGBoost, w którym chcemy znaleźć interakcje oraz zbiór danych, który zastał użyty do wyuczenia modelu, zapisany w postaci `data.table`. Funkcja ta generuje tabelę ważności interakcji lub par znajdujących się w modelu. Zawiera ona sumaryczną wartość Gain oraz częstotliwość wystąpień danej interakcji lub pary (Tabela 3.1). Tabela ta jest obiektem klasy *interactions*.

Wynikiem funkcji `plot`, dla argumentu klasy *interactions*, jest wykres przedstawiony na Rysunku 4.2. Jest to wykres kafelkowy, który zawiera wszystkie istniejące pary zmiennych które są w interakcji. Na osi y są zmienne które w jakiejkolwiek interakcji pełnią funkcję rodzica. Natomiast na osi x zmienne które pełnią funkcję dziecka w co najmniej jednej interakcji. Jeżeli między dwoma zmiennymi zachodzi interakcja to na skrzyżowaniu prostych przechodzących przez te zmienne jest wypełniony kwadrat, w przeciwnym przypadku jest pu-



Rysunek 3.2: Wynik funkcji `plot` z obiektem *interactions* użytym jako argument oraz z domyślnymi parametrami. Wykres przedstawia wszystkie interakcje znalezione w modelu. Im ciemniejszy kolor kratki na przecięciu dwóch zmiennych tym interakcja między tymi zmiennymi jest bardziej istotna.

ste pole. Kolor kwadratu wskazuje na siłę interakcji, która jest sumaryczną wartością Gain dla wszystkich wystąpień danej interakcji. Zakres wartości Gain dla większej czytelności i łatwiejszego użytkowania pakietu został podzielony na cztery równe części: `very low, low, medium, high`. Dzięki zastosowaniu czterech równych interwałów, a nie np. kwantylowego podziału możemy wyodrębnić interakcje, które rzeczywiście są najbardziej istotne w modelu.

W pakiecie istnieje możliwość badania wszystkich występujących par zmiennych. Aby uzyskać analogiczny do rysunku 3.2 wykres dla par występujących w modelu należy wywołać funkcję *interactions* z dodatkowym argumentem: `option = "pairs"`, a następnie powstały obiekt wstawić jako argument funkcji `plot`.

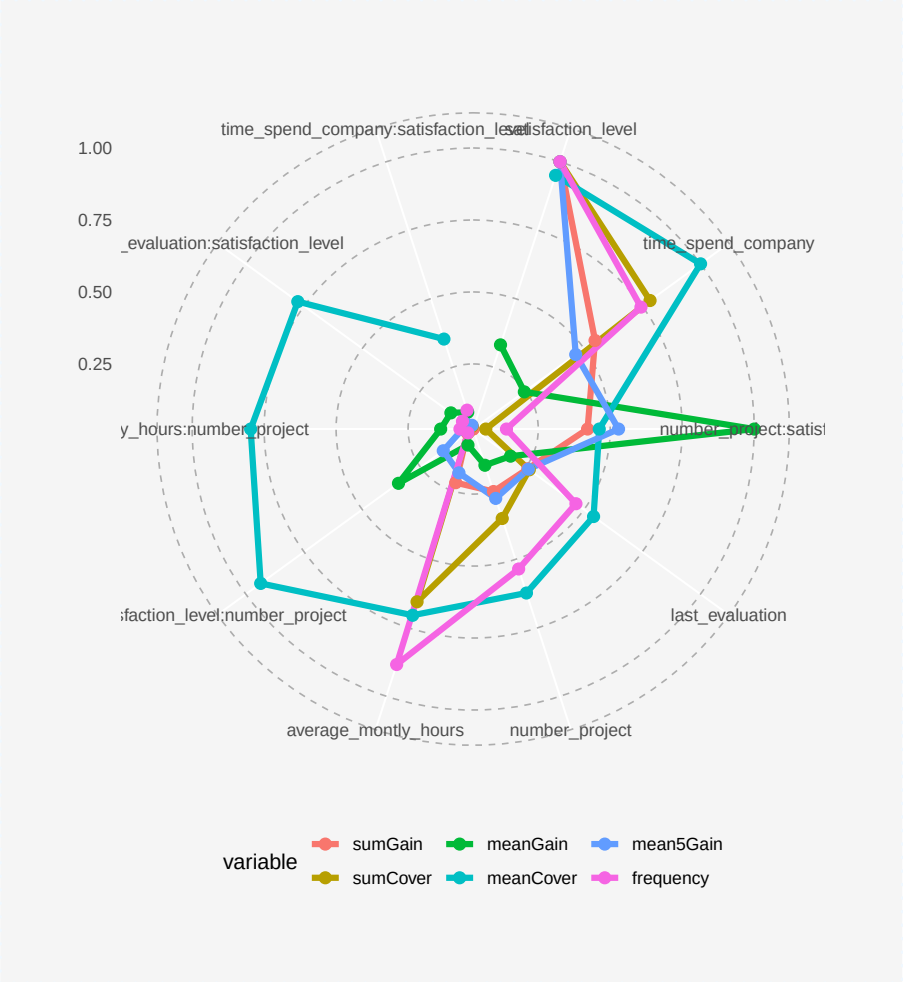
Tablica 3.1: Wynik funkcji `interactions` z pakietu `EIX`, dla modelu XGBoost na danych dotyczących odejść pracowników z firmy.

	Parent	Child	sumGain	frequency
1	last_evaluation	average_montly_hours	745.59	2
2	last_evaluation	satisfaction_level	708.87	4
3	last_evaluation	time_spend_company	635.00	3
4	satisfaction_level	time_spend_company	560.00	2
5	last_evaluation	number_project	390.19	1
6	average_montly_hours	time_spend_company	318.01	2
7	average_montly_hours	last_evaluation	312.11	2
8	average_montly_hours	number_project	174.18	1

3.3. Ważność zmiennych

Tabelę ważności zmiennych zawierającą miary zmiennych przedstawione w rozdziale 2.1, można uzyskać za pomocą funkcji `importance` z następującymi argumentami: model XGBoost, zbiór danych.

Tabelę tę w zależności od parametru `option = "interactions"`, `"variables"`, `"both"` można wywołać odpowiednio dla samych interakcji, dla pojedynczych zmiennych oraz jednocześnie dla interakcji i pojedynczych zmiennych (ustawienie domyślne). Warto zauważyć, że ostatnia opcja nie jest połączeniem dwóch pierwszych opcji w jedną tabelkę. W opcji `"both"` zmienne, które są dziećmi w interakcji, nie są brane pod uwagę przy obliczaniu miar ważności dla pojedynczych zmiennych. Dlatego wartości różnych miar dla tych samych pojedynczych zmiennych w opcji `"variables"` i `"both"` mogą się różnić. Wszystkie tabele będące wynikiem funkcji `importance` są uporządkowane zgodnie z malejącą wartością miary sumGain. Przykładowy wynik tej funkcji jest przedstawiony w tabeli 3.2.



Rysunek 3.3: Wykres ważności zmiennych dla 10 najważniejszych zmiennych i interakcji zmiennych wygenerowany za pomocą funkcji `plot` z obiektem klasy *importance*, jako argumentem i z domyślnymi parametrami. Wykres przedstawia sześć miar ważności zmiennych: **sumGain**, **meanGain**, **mean5Gain**, **sumCover**, **frequency**.

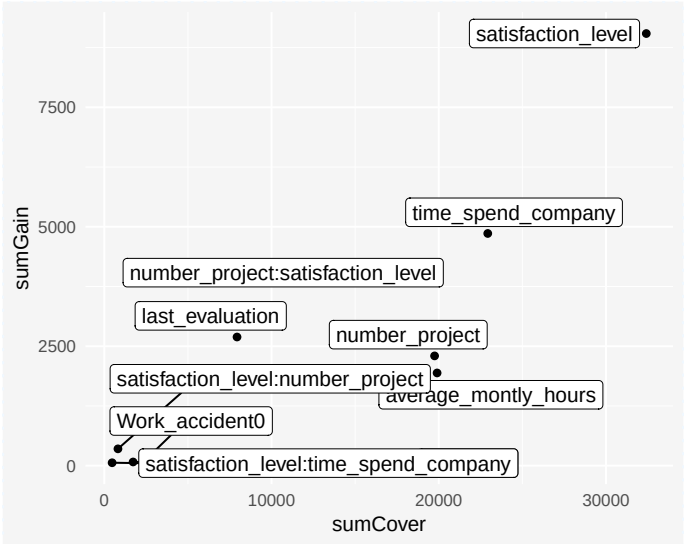
Tablica 3.2: Wynik funkcji `importance` z pakietu `EIX`, dla modelu XGBoost na danych dotyczących odejść pracowników z firmy.

Feature	sumGain	sumCover	meanGain	meanCover	frequency	mean5Gain
satisfaction_level	10040.00	43920.00	264.10	1156.00	38.00	1513.00
time_spend_company	4016.00	19820.00	267.70	1321.00	15.00	670.40
number_project	3706.00	13940.00	264.70	995.60	14.00	697.40
last_evaluation	1181.00	15340.00	90.81	1180.00	13.00	183.00
average_montly_hours	886.00	18190.00	46.63	957.60	19.00	97.50
last_evaluation:average_montly_hours	745.60	1767.00	372.80	883.70	2.00	372.80
last_evaluation:satisfaction_level	708.90	2985.00	177.20	746.20	4.00	177.20
last_evaluation:time_spend_company	635.00	2535.00	211.70	845.00	3.00	211.70

Wykres miar ważności zmiennych można uzyskać za pomocą funkcji `plot` z wynikiem funkcji `importance`, jako argumentem. Wynikiem tej funkcji dla domyślnych paramentów jest wykres radarowy, który po zewnętrznej stronie okręgu zawiera nazwy zmiennych lub interakcji, natomiast kolorowe łamane oznaczają poszczególne miary ważności zmiennych. Wszystkie miary zostały przeskalowane, w taki sposób, aby zawierały wartości od 0 do 1. Użytkownik może zdefiniować liczbę pozycji, które mają być przedstawione na wykresie za pomocą parametru `top`. W zależności od postaci obiektu *importance* istnieje również możliwość stworzenia trzech rodzajów wykresów przedstawiających pojedyncze zmienne, interakcje lub jednocześnie pojedyncze zmienne i interakcje.

Rysunek 3.3 jest wynikiem funkcji `plot`, z domyślnym parametrem `radar = TRUE`. Zawiera zarówno pojedyncze zmienne, jak i interakcje. Poszczególne pozycje są uporządkowane malejąco pod względem miary sumGain. Zmienna, która ma najwyższy wskaźnik sumGain znajduje się najbliżej pozycji godziny 12 – z prawej strony. Zmienne o niższym poziomie sumGain ułożone są zgodnie z kierunkiem ruchu wskazówek w kolejności malejącej.

Za pomocą funkcji `plot` z obiektem klasy *importance* jako argumentem i z parametrem `radar = FALSE` można dokładnie porównać dwie wybrane miary zmiennych ustawiając jako parametry `xlab` i `ylab` (Rysunek 3.3). Wybrane zmienne znajdują się odpowiednio na osi x



Rysunek 3.4: Wynik funkcji `plot` z obiektem klasy *importance*, jako argumentem i z parametram `radar = FALSE`, przedstawiający dwie miary ważności zmiennych: **sumCover** na osi x i **sumGain** na osi y.

i osi y. Funkcja analogicznie do powyższych funkcji dotyczących analizy ważności zmiennych ma możliwość przedstawienia na wykresie w zależności od postaci obiektu klasy *importance*, pojedynczych zmiennych, samych interakcji oraz zarówno pojedynczych zmiennych i interakcji. Liczbę elementów widocznych na wykresie można regulować za pomocą parametru `top`.

3.4. Analiza predykcji pojedynczej obserwacji

Do analizy predykcji pojedynczej obserwacji wykonanej za pomocą modelu XGBoost służy funkcja `waterfall`. Niezbędne argumenty tej funkcji to model XGBoost, zbiór danych oraz numer obserwacji, dla której predykcja modelu ma zostać wyjaśniona.

W funkcji `waterfall` został wykorzystany, za zgodą autora - David'a Foster'a - kod zawarty w pakiecie `xgboostExplainer` [15, 16]. Pakiet ten to narzędzie do wykonywania analizy predykcji modelu XGBoost dla pojedynczej obserwacji. Pakiet `EIX` rozszerza tą analizę przez dodanie interakcji, które mają wpływ na predykcję pojedynczej obserwacji.

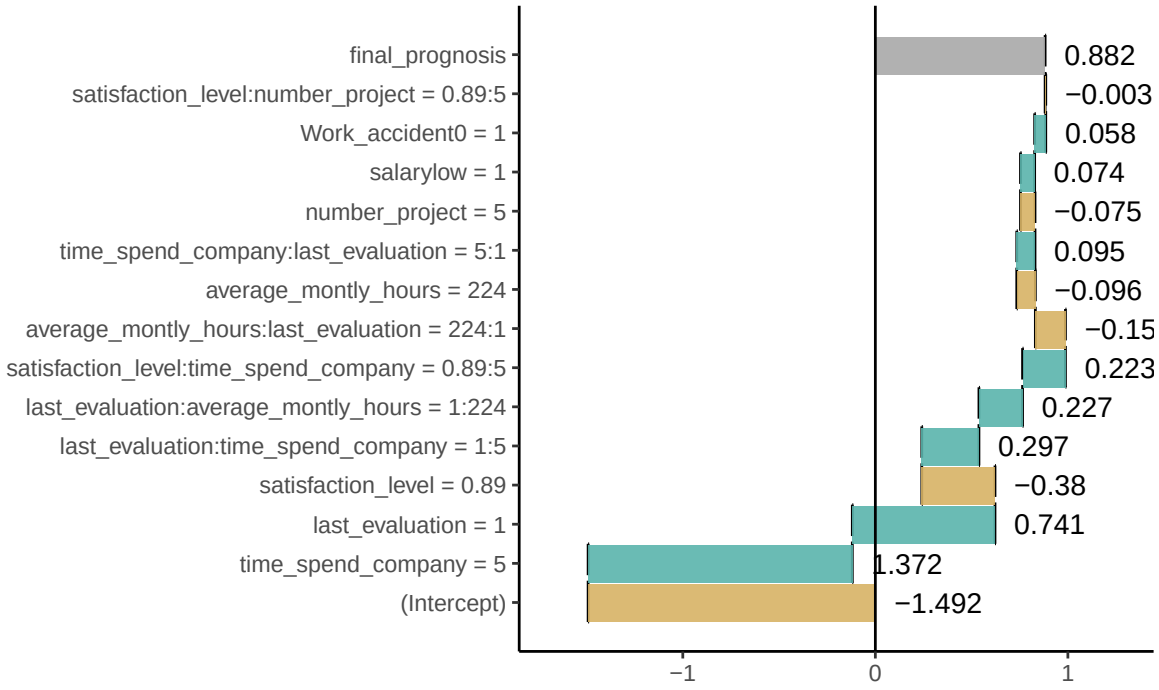
Funkcja ta korzysta również z metodologii pochodzącej z pakietu `breakDown` [42]. Wewnątrz funkcji tworzony jest obiekt klasy `broken`, a następnie jest możliwość stworzenia wykresu za pomocą funkcji `plot` zdefiniowanej w pakiecie `breakDown`. Obiekt `broken` to tabela zawierająca wpływ poszczególnym zmiennych i interackcji na predykcję określonej obserwacji (Tabela 3.3). W zależności od paranetru `opt` istnieje możliwość stworzenia tabeli uwzględniającej interakcje (parametr `option = "interactions"`) oraz nie uwzględniającej ich (`option~=~"variables"`).

Rysunki 3.5 i 3.6 przedstawiają wyjaśnienia predykcji dla przykładowej obserwacji. Na osi y znajdują się kolejno od dołu :

- `intercept`, prawdopodobieństwo, że losowo wybrana obserwacja będzie przyjmować wartość 1. W danych jest ok. 24% obserwacji dla których zmienna `left` wynosi 1.
- nazwy zmiennych, które miały wpływ na predykcje, ułożone rosnąco pod względem wielkości wpływu na predykcje,
- `final_prognosis` - ostateczna predykcja modelu.

Na osi x znajduje się log-szansa wybranej obserwacji. Każdy prostokąt oznacza wpływ na predykcję tej obserwacji (im większy prostokąt, tym większy wpływ na predykcję). Zmienne, których sumaryczny wpływ na predykcję jest ujemny są oznaczone kolorem beżowym. W tym przypadku oznacza to, że zmienne te zmniejszają prawdopodobieństwo odejścia pracownika z firmy. Prostokąty oznaczone kolorem morskim symbolizują zmienne, które podwyższają predykcje badanej obserwacji, czyli zwiększają prawdopodobieństwo odejścia pracownika z firmy. Kolorem szarym oznaczona jest końcowa predykcja. Liczby przy prostokątach oznaczają wartość log-szansy, która opisuje wpływ na predykcje obserwacji danej zmiennej.

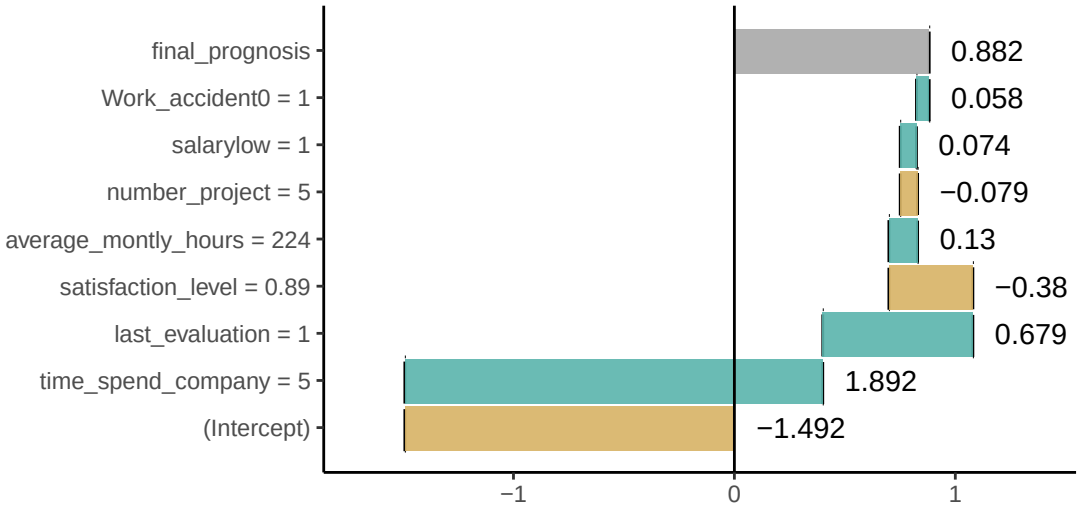
Istnieje możliwość stworzenia wykresu uwzględniającego interakcje przy użyciu obiektu `broken`, stworzonego z parametrem `option = "interactions"` (Rysunek 3.5) oraz nie uwzględniające ich `option = "variables"` (Rysunek 3.6).



Rysunek 3.5: Wyjaśnienie predykcji wybranej obserwacji ze zbioru danych o odejściach pracowników z firmy z uwzględnieniem interakcji zachodzących w modelu. Wykres został wykonany za pomocą pakietu `EIX` przy użyciu funkcji `plot`, z obiektem klasy `broken`, jako argumentem, z domyślnymi parametrami

Tablica 3.3: Wynik funkcji `waterfall` z pakietu `EIX`, dla modelu XGBoost na danych dotyczących odejść pracowników z firmy.

	contribution
(Intercept)	-1.492
time_spend_company = 5	1.372
last_evaluation = 1	0.741
satisfaction_level = 0.89	-0.380
last_evaluation:time_spend_company = 1:5	0.297
last_evaluation:average_montly_hours = 1:224	0.227
satisfaction_level:time_spend_company = 0.89:5	0.223
average_montly_hours:last_evaluation = 224:1	-0.156
average_montly_hours = 224	-0.096
time_spend_company:last_evaluation = 5:1	0.095
number_project = 5	-0.075
salarylow = 1	0.074
Work_accident0 = 1	0.058
satisfaction_level:number_project = 0.89:5	-0.003
final_prognosis	0.882
baseline:	0



Rysunek 3.6: Wyjaśnienie predykcji wybranej obserwacji ze zbioru danych o odejściach pracowników z firmy bez uwzględnienia interakcji zachodzących w modelu. Wykres został wykonany za pomocą pakietu `EIX` przy użyciu funkcji `plot`, z obiektem klasy `broken`, jako argumentem, z parametrem `option = "variables"`.

Rozdział 4

Przykład zastosowania biblioteki EIX

4.1. Dane i model

Zbiór danych, który został użyty do wygenerowania przykładów, pochodzi ze strony www.kaggle.com i jest to zbiór symulowany. Zawiera 14999 obserwacji reprezentujących pracowników firmy. Każda obserwacja opisana jest przez 10 kolumn zawierających następujące informacje:

- `satisfaction_level` (zmienna ciągła) - poziom satysfakcji. Zakres od 0.09 do 1, w czym prawie 10200 pracowników zdeklarowało poziom satysfakcji powyżej 0.5. Średnia: 0.613.
- `last_evaluation` (zmienna ciągła) - ostatnia ocena wydajności. Zakres: od 0.36 do 1.
- `number_project` (zmienna dyskretna) - liczba projektów, w których pracownik brał udział. Zakres: 2 - 7.
- `average_montly_hours` (zmienna ciągła) - średnia miesięczna liczba godzin spędzonych w pracy. Średnia dla wszystkich pracowników: ok. 200 h miesięcznie.
- `time_spend_company`(zmienna dyskretna) - liczba lat przepracowanych w firmie. Zakres: 2-10, średnia: ok.3,5 roku.
- `Work_accident` (zmienna binarna) - informacja, czy pracownik miał kiedykolwiek wypadek w miejscu pracy. Pracowników którzy mieli wypadek w pracy jest 2 169.
- `left` (zmienna binarna) - informacja, czy pracownik opuścił firmę. W danych jest 3 571 pracowników, którzy opuścili firmę i 11 428 którzy w niej pozostali.
- `promotion_last_5years` (zmienna binarna) - informacja, czy pracownik miał awans w ciągu ostatnich 5 lat. W danych jest 319 pracowników, którzy otrzymali awans w ciągu ostatnich 5 lat
- `sales` (zmienna kategoryczna) - nazwa działu, w którym pracował. Dziesięć możliwych działów: `sales, technical, support, IT, product_mng, marketing, RandD, hr, accounting, management`.
- `salary` (zmienna kategoryczna) - poziom pensji. Trzy możliwe poziomy: niski, średni, wysoki - `low, medium, high`.

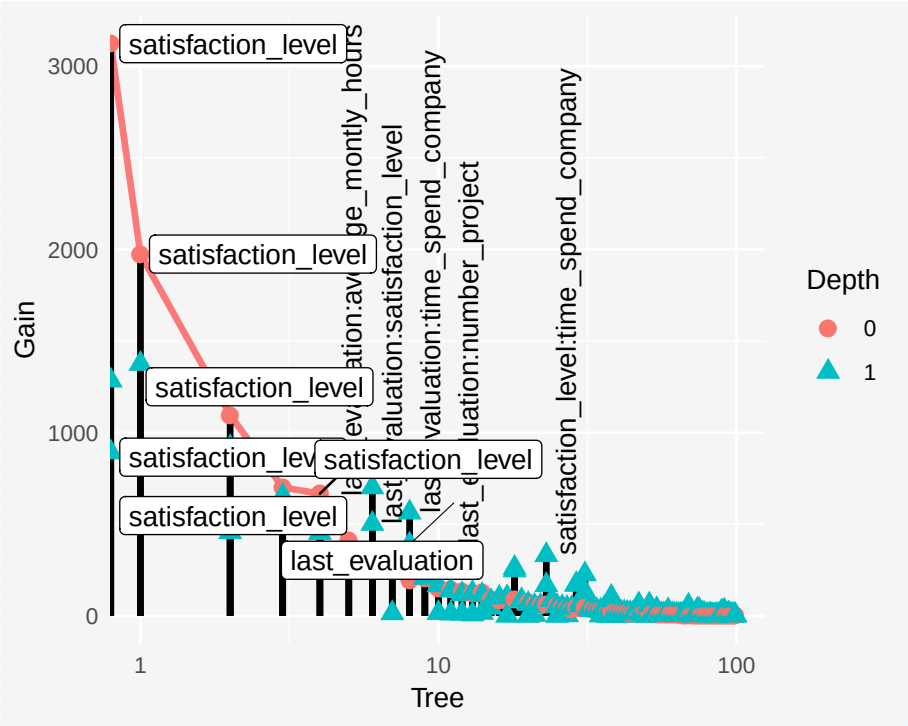
W poniższym przykładzie omówione będzie zadanie klasyfikacji. Interesuje nas predykcja odejścia pracownika z pracy, więc zmienną objaśnianą będzie kolumna `left`, a pozostałe kolumny będą zmiennymi objaśniającymi.

Za pomocą funkcji `xgboost` [14] został stworzony model XGBoost składający się ze 100 drzew o maksymalnej głębokości 2, za co odpowiadają następujące parametry w funkcji: `nrounds` i `max_depth`. Przed stworzeniem modelu dane zostały odpowiednio przygotowane - zmienne kategoryczne zostały odpowiednio zakodowane za pomocą funkcji `as.factor()`. Jest to niezbędne, gdyż domyślnie funkcja `xgboost` traktuje wszystkie cechy jako zmienne ciągłe. Następnie została stworzona macierz rzadka, która zmienne zdefiniowane jako `factor` koduje zgodnie z metoda *one-hot-encoding*.

4.2. Analiza modelu

Badanie modelu warto rozpocząć od wygenerowania wykresu *lollipop* (Rysunek 4.1). Analiza tego wykresu pozwala zidentyfikować najważniejsze zmienne i interakcje w modelu.

W badanym przypadku można zauważyć, że zmienna `satisfacion_level` pełni ważną rolę w tym modelu. Często w nim występuje i osiąga wysoką wartość Gain. Jest to zgodne z intuicją, gdyż poziom satysfakcji pracownika powinien mieć największy wpływ na jego ewentualne odejście. Kolejną ważną zmienną jest `last_evaluation`. Wskazuje ona, że ostatnia ocena pracownika, ma istotny wpływ na predykcję odejścia, co także jest logicznym wnioskiem.



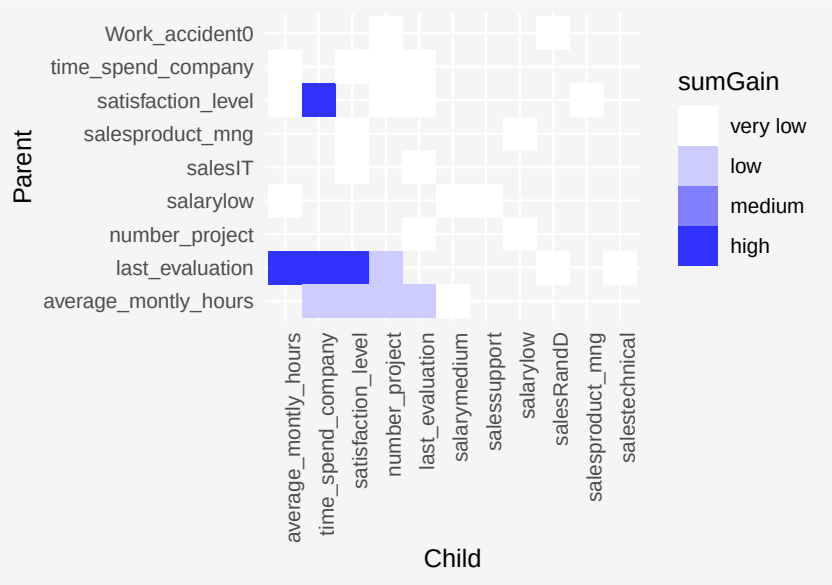
Rysunek 4.1: Wizualizacja modelu w postaci wykresu *lollipop* z domyślnymi parametrami dla danych dotyczących odejścia pracowników z firmy. Na wykresie widoczne istotne zmienne i interakcje m.in: `satisfacion_level`, `last_evaluation`, `last_evaluation:satisfacion_level`, `last_evaluation:average_montly_hours`.

Z wykresu 4.1 można też odczytać, że ważnymi interakcjami są: `last_evaluation:satisfacion_level`, `last_evaluation:average_montly_hours`, `last_evaluation:time_spend_company`, `last_evaluation:number_project` , `satisfaction_level:time_spend_company`.

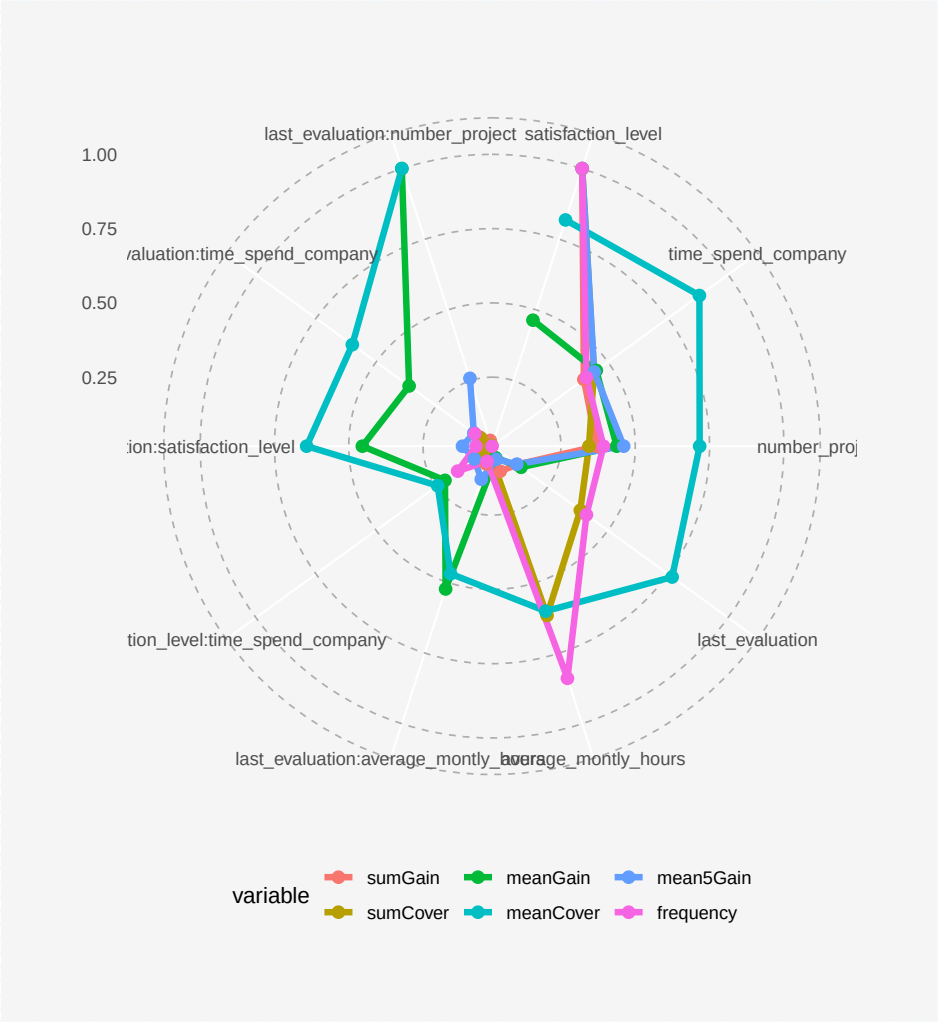
Zmienna opisująca ostatnią ocenę pracownika jest w interakcji z następującymi zmiennymi: średnia liczba godzin spędzonych w pracy miesięcznie, poziom satysfakcji, liczba lat przepracowanych w firmie, liczba zrealizowanych projektów. Z doświadczenia wiadomo, ze cechy te mają wpływ na ocenę pracownika przez pracodawcę. Jedynie interakcja między oceną pracownika, a poziomem satysfakcji może budzić wątpliwości, jednakże pracownik który jest zadowolony ze swojej pracy na ogół lepiej ją wykonuje, co może wpływać na ocenę pracodawcy. Zmienne te są ze sobą powiązane, dlatego ich korelacja ma wpływ na predykcję odejścia pracownika z firmy.

Analogicznie interakcja między zmiennymi opisującymi poziom satysfakcji i liczbą lat przepracowanych w firmie jest również racjonalna np. pracownicy, którzy przepracowali wiele lat w tej samej firmie mogą być znudzeni swoim obecnym zajęciem, dlatego są mniej nią usatysfakcjonowani i chętniej opuszczają firmę.

Wykres 4.2 przedstawia wszystkie interakcje zachodzące w modelu. Jego analiza pozwala wyodrębnić interakcje, które były widoczne we wstępnej fazie pracy z modelem, ale można również dostrzec mniej istotne interakcje, takie jak:
`last_evaluation:number_project`, `average_montly_hours:time_spend_company`, `average_montly_hours:satisfaction_level`, `average_montly_hours:number_project`, `average_montly_hours:last_evaluation`.



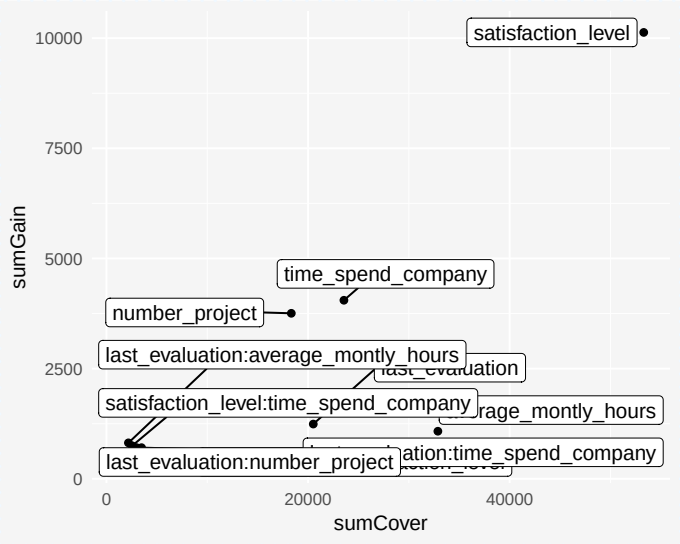
Rysunek 4.2: Wykres *interactions* z domyślnymi parametrami obrazujący interakcje zachodzące w badanym modelu. Na wykresie widoczne wszystkie interakcje znalezione w modelu. Wśród nich najistotniejsze są: `last_evaluation:satisfacion_level`, `last_evaluation:average_montly_hours`, `last_evaluation:time_spend_company`, `last_evaluation:number_project` , `satisfaction_level:time_spend_company`.



Rysunek 4.3: Wykres radarowy wygenerowany z obiektu *importance*, za pomocą funkcji `plot` z domyślnymi parametrami przedstawiający 10 najważniejszych zmiennych i interakcji w modelu. Wśród nich pięć interakcji.

Tablica 4.1: Ranking ważności zmiennych i interakcji w modelu.

Feature	sumGain	sumCover	meanGain	meanCover	frequency	mean5Gain
satisfaction_level	10130.00	53300.00	180.90	951.70	56.00	1513.00
time_spend_company	4052.00	23560.00	176.20	1024.00	23.00	670.40
number_project	3756.00	18340.00	170.70	833.40	22.00	697.40
last_evaluation	1243.00	20520.00	54.06	892.10	23.00	183.00
average_montly_hours	1080.00	32880.00	22.98	699.60	47.00	97.50
last_evaluation:average_montly_hours	816.30	2183.00	204.10	545.70	4.00	204.10
satisfaction_level:time_spend_company	749.80	2553.00	83.31	283.70	9.00	138.30
last_evaluation:satisfaction_level	708.90	2985.00	177.20	746.20	4.00	177.20
last_evaluation:time_spend_company	707.70	3483.00	141.50	696.50	5.00	141.50
last_evaluation:number_project	390.20	1163.00	390.20	1163.00	1.00	390.20
average_montly_hours:time_spend_company	368.30	1546.00	92.08	386.50	4.00	92.08
average_montly_hours:last_evaluation	333.80	1542.00	111.30	514.00	3.00	111.30
Work_accident0	243.70	5506.00	34.82	786.60	7.00	47.49
average_montly_hours:number_project	224.10	1403.00	56.03	350.80	4.00	56.03
average_montly_hours:satisfaction_level	214.00	1037.00	71.34	345.60	3.00	71.34



Rysunek 4.4: Wykres ważności zmiennych i interakcji z **sumCover** na osi x i **sumGain** na osi y.

Wykres radarowy przedstawiony na rysunku 4.3 pokazuje najważniejsze zmienne i interakcje w modelu. W przypadku badanego modelu najważniejszą zmienną zgodnie z wcześniejszą analizą jest `satisfaction_level`, następnie `time_spend_company`, `number_project`, `last_evaluation`, `average_montly_hours`, a od szóstego do dziesiątego miejsca w rankingu znajdują się następujące interakcje: `last_evaluation:average_montly_hours`, `satisfaction_level:time_spend_company`, `last_evaluation:satisfaction_level`, `last_evaluation:time_spend_company`, `last_evaluation:number_project`. Warto zauważyć, że pięć interakcji w tym modelu znajdują się w pierwszej 10 najważniejszych zmiennych. Fakt ten podkreśla istotność badania interakcji w przypadku tego modelu.

Wykres 4.4 przedstawia wykres `importance` dla analizowanego modelu. Na osi x znajduje się miara sumCover, na osi y znajduje się sumGain. Interpretacja tego wykresu potwierdza poprzednie wnioski dotyczące najistotniejszych zmiennych w badanym modelu.

Dokładny ranking interakcji i różne miary ważności można są widoczne w tabeli 4.1 będącej wynikiem funkcji `importance`.

4.3. Analiza predykcji wybranej obserwacji

Do przykładu została wybrana obserwacja o numerze 9 w prezentowanym zbiorze danych. Wartości zmiennych dla tej obserwacji zostały przedstawione w tabeli 4.2.

Na wykresie 4.5 zmienna `time_spend_company` ma największy wpływ na ostateczną predykcję. Analizowany pracownik w firmie pracował 5 lat, czyli powyżej średniej, ostatnia ocena (`last_evaluation`) wynosiła 1, czyli maksymalną wartość, a średnia liczba godzin, jaką spędzał ten pracownik w pracy (`average_montly_hours`) wynosiła 224 - również jest to wartość powyżej średniej. Te trzy zmienne w największym stopniu przyczyniły się według modelu do odejścia pracownika z firmy, co wydaje się mieć racjonalne uzasadnienie. Pracownik, który pracował w firmie dłużej niż inni pracownicy, może być znudzony wykonywaną pracą, która nie jest już dla niego ambitnym wyzwaniem - otrzymuje maksymalną ocenę wydajności. Liczba przepracowanych godzin, która jest powyżej średniej, również może skłaniać pracownika do odejścia. Zmienna, która w największym stopniu obniża predykcję to `satisfaction_level`.

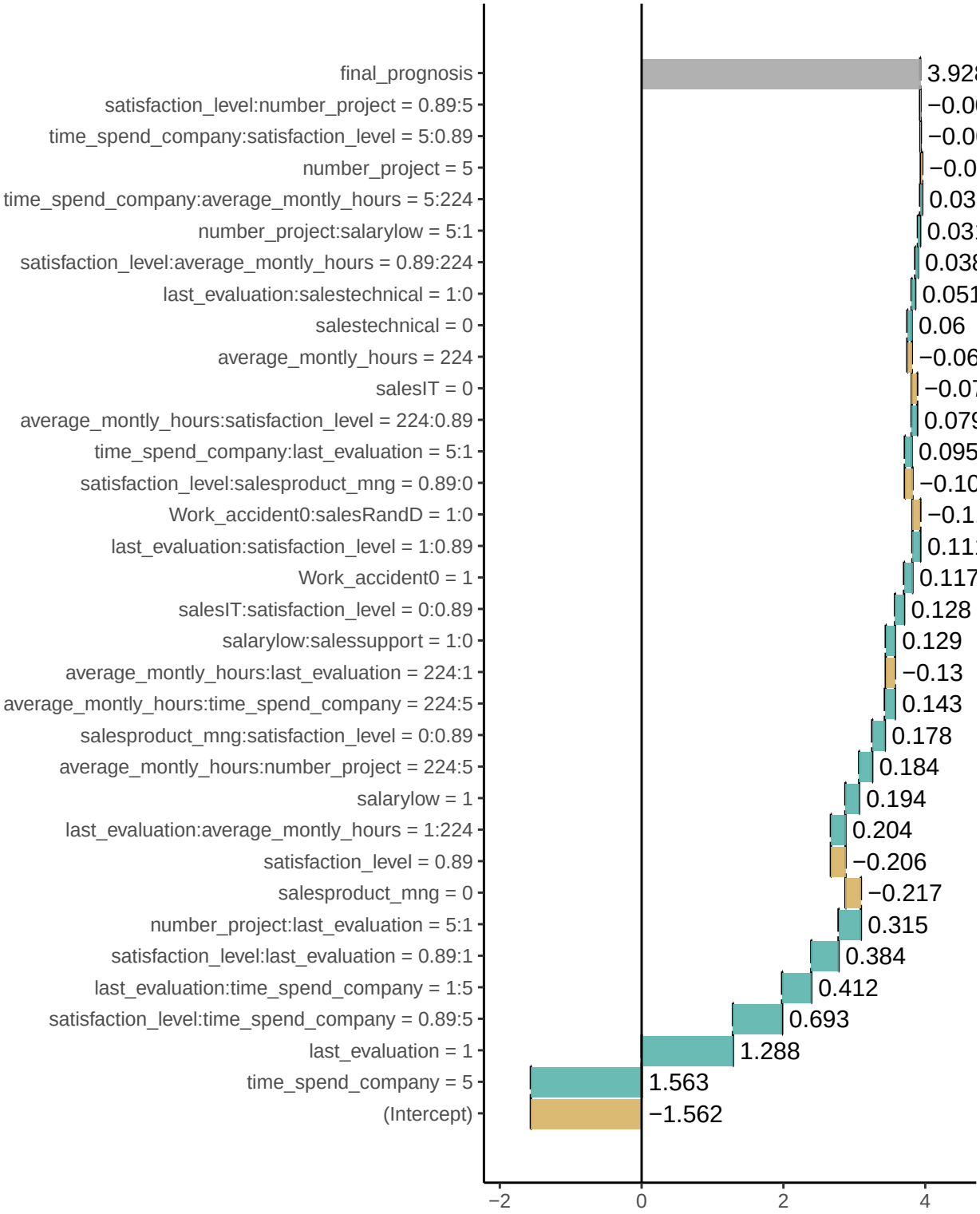
Tablica 4.2: Wartości wszystkich zmiennych dla obserwacji numer 9, której predykcja jest przedstawiona na wykresie 4.5. Rzeczywista wartość przewidywanej zmiennej (`left`) wynosi 1.

satisfaction_level	0.89
last_evaluation	1
number_project	5
average_montly_hours	224
time_spend_company	5
Work_accident	0
promotion_last_5years	0
sales	sales
salary	low
left	1

Jest to uzasadnione, gdyż wartość tej zmiennej wynosi 0.89, czyli powyżej średniej. Pozostałymi zmiennymi, które miały nieznaczny wpływ na badaną obserwację były: liczba zrealizowanych projektów (`number_project`), wysokość pensji (`salary`) i wskaźnik opisujący, czy badany doznał wypadku w pracy (`Work_accident`).

Wykres 4.5 przedstawia wyjaśnienie predykcji dla tej samej obserwacji z uwzględnieniem interakcji. Wpływ niektórych pojedynczych zmiennych obniżył się, gdyż część ich wartości została uznana za wpływ interakcji. Ponownie z analizy wykresu wynika, że najważniejszymi czynnikami decydującymi o opuszczeniu przez pracownika firmy są zmienne `time_spend_company` i `last_evaluation`. Często znajdują się również w interakcjach uwzględnionych w modelu, co jeszcze bardziej podkreśla ich znaczenie w predykcji badanej obserwacji.

Hipotetyczny pracodawca, który dokonałby powyższej analizy, z wykresów mógłby wywnioskować, że aby zmienić potencjalną decyzję pracownika powinien zmienić ostatnią ocenę jego wydajności. Nie ma on niestety wpływu na zmienną mającą największy wpływ na predykcję, czyli staż pracy pracownika.



Rysunek 4.5: Wyjaśnienie predykcji badanego modelu dla obserwacji numer 9. Na wykresie uwzględnione interakcje zachodzące w modelu. Zmiennymi mającymi największy wpływ na predykcje są: `time_spend_company`, `last_evaluation`, `satisfaction_level:time_spend_company`, `last_evaluation:time_spend_company`.

Podsumowanie

Celem pracy magisterskiej była analiza struktury i ekstrakcja wiedzy z klasyfikatorów otrzymanych metodą XGBoost z uwzględnieniem interakcji. W pierwszym rozdziale zostały opisane drzewa decyzyjne, komitety klasyfikatorów, a także wprowadzono modele XGBoost i LightGBM. Opisano w nim również najważniejsze algorytmy optymalizujące użyte w pakietach, w których zaimplementowane są wyżej wymienione metody. W drugim rozdziale opisane zostały miary ważności zmiennych użytych w pakiecie `EIX`, a także sposoby uwzględniania interakcji przez model XGBoost oraz metodologia wykorzystana do analizy predykcji pojedynczej obserwacji. W kolejnym rozdziale zaprezentowano pakiet `EIX`, a następnie pokazano przykład użycia tego pakietu na danych dotyczących odejść pracowników z firmy.

Obecnie dostępnych jest co najmniej kilka narzędzi zajmujących się tematyką analizy struktury modeli XGBoost. W pakiecie `xgboost` istnieje funkcja która bada podstawowe miary ważności zmiennych, a pakiet `xgboostExplainer` to narzędzie do wykonywania analizy pojedynczej predykcji. W pakiecie `EIX` wykorzystano dostępną wiedzę - udoskonalono już istniejące, a także stworzono nowe funkcjonalności przydatne w analizie struktury modelu XGBoost. Przede wszystkim narzędzie to znajduje interakcje uwzględnione w modelu. Zdecydowana większość funkcji dostępnych w pakiecie wspiera również modele otrzymane metodą LightGBM. Warto zwrócić uwagę na fakt, iż narzędzie prezentowane w tej pracy posiada wiele funkcji wizualizujących wyniki, co jest niewątpliwie jego zaletą.

Możliwości pakietu `EIX` zostały pokazane na podstawie interpretacji modelu XGBoost, którego zadaniem była klasyfikacja odejść pracowników z firmy. Za pomocą pakietu zostały wyodrębnione zmienne, które były najistotniejsze w modelu oraz interakcje w nim uwzględnione. W badanym przypadku największy wpływ na predykcję modelu miał poziom satysfakcji pracownika i jego ostatnia ocena wydajności. Za pomocą opisywanego pakietu zostały znalezione najważniejsze zależności w modelu. Ostatnia ocena pracownika jest w interakcji z następującymi zmiennymi: liczbą godzin przepracowanych w miesiącu, poziomem satysfakcji oraz liczbą lat przepracowanych w firmie. Warto podkreślić, że wymienione zależności znalazły się w rankingu 10 najważniejszych zmiennych i interakcji mających wpływ na model. Ukazuje to istotność badania interakcji w modelu.

Z pomocą pakietu `EIX` dokonano też przykładowej analizy predykcji pojedynczej obserwacji. Analiza ta pozwala zweryfikować działanie modelu. Jest szczególnie istotna, w przypadkach gdy predykcja każdej pojedynczej obserwacji nie może być przeprowadzona w błędny sposób np.: w medycynie, gdy obserwacją jest pacjent. Dodatkowo informacja o tym, jakie zmienne mają największy wpływ na predykcję, pomaga decydować, np. które symptomy choroby u danego pacjenta likwidować w pierwszej kolejności.

Z pewnością modele uczenia maszynowego będą stawać się coraz bardziej skomplikowane, dlatego rozwój metod i pakietów wyjaśniających te modele takich jak `EIX` jest konieczny. Niezrozumienie działania modelu, może doprowadzić do faworyzowania cech nieistotnych jak i pomijania cech istotnych. W obu przypadkach, doprowadzić to może do błędnej interpretacji wyniku końcowego. Dlatego też, bez pomocy pakietów, takich jak stworzony w tej pracy,

poprawne korzystanie ze złożonych modeli uczenia maszynowego będzie trudne, a w niektórych przypadkach zupełnie niemożliwe.

Bibliografia

[1] Dan Apley. *ALEPlot: Accumulated Local Effects (ALE) Plots and Partial Dependence (PD) Plots.* R package version 1.1, 2018. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=ALEPlot>.

[2] Michel Ballings, Dirk Van den Poel. *rotationForest: Fit and Deploy Rotation Forest Models.* R package version 0.1.3, 2017. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=rotationForest>.

[3] Przemysław Biecek. *DALEX: Explainers for Complex Predictive Models in R.* Journal of Machine Learning Research, 2018, 19 (84), pp. 1-5. [dostęp: 1 marca 2019], <http://jmlr.org/papers/v19/18-416.html>.

[4] Przemysław Biecek. *Przewodnik po pakiecie R 4.0.* [dostęp: 12 listopada 2018], <http://pbiecek.github.io/Przewodnik/index.html>.

[5] Przemyslaw Biecek. *Package DALEX.* [dostęp: 12 listopada 2018], <https://github.com/pbiecek/DALEX>.

[6] Przemyslaw Biecek, Aleksandra Grudziąż. *Package breakDown.* [dostęp: 12 listopada 2018], <https://github.com/pbiecek/breakDown>.

[7] Houtao Deng. *Guided Random Forest in the RRF Package.* arXiv:1306.0237, 2013.

[8] Houtao Deng, George Runger. *Gene Selection with Guided Regularized Random Forest.* Pattern Recognition, 2013, 46(12), pp. 3483-3489.

[9] Houtao Deng, George Runger. *Feature Selection via Regularized Trees.* The 2012 International Joint Conference on Neural Networks (IJCNN), 2012.

[10] Albert Cheng. *lightgbmExplainer: An R package that makes lightgbm models fully interpretable.* R package version 0.1.0.

[11] Tianqi Chen, Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System.* Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp.785–794.

[12] Tianqi Chen,Tong He,Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan Xie, Min Lin, Yifeng Geng, Yutian Li, XGBoost contributors. *Package 'xgboost'.* Dokumentacja pakietu xgboost, 2018. [dostęp: 12 listopada 2018], <https://cran.r-project.org/web/packages/xgboost/xgboost.pdf>.

[13] Tianqi Chen,Tong He,Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan Xie, Min Lin, Yifeng Geng, Yutian Li, XGBoost contributors. *Package xgboost.* [dostęp: 12 listopada 2018], <https://github.com/dmlc/xgboost/tree/master/R-package>.

[14] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan Xie, Min Lin, Yifeng Geng and Yutian Li. *xgboost: Extreme Gradient Boosting.* R package version 0.81.0.1, 2019. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=xgboost>.

[15] David Foster. *NEW R package that makes XGBoost interpretable.* Medium, 27 września 2017. [dostęp: 12 listopada 2018], <https://medium.com/applied-data-science/new-r-package-the-xgboost-explainer-51dd7d1aa211>.

[16] David Foster. *xgboostExplainer: XGBoost Model Explainer.* R package version 0.1, 2017. [dostęp: 12 listopada 2018], <https://github.com/AppliedDataSciencePartners/xgboostExplainer>.

[17] Brandon M. Greenwell. *pdp: An R Package for Constructing Partial Dependence Plots.* The R Journal, 2017, 9(1), 421–436. [dostęp: 1 marca 2019], <https://journal.r-project.org/archive/2017/RJ-2017-016/index.html>.

[18] Thomas Grubinger, Achim Zeileis, Karl-Peter Pfeiffer. *evtree: Evolutionary Learning of Globally Optimal Classification and Regression Trees in R.* Journal of Statistical Software, 2014, 61(1), 1-29. [dostęp: 1 marca 2019], <http://www.jstatsoft.org/v61/i01/>.

[19] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2016.

[20] Torsten Hothorn, Kurt Hornik, Achim Zeileis. *Unbiased Recursive Partitioning: A Conditional Inference Framework.* Journal of Computational and Graphical Statistics, 2006, 15(3), pp. 651-674.

[21] Torsten Hothorn, Achim Zeileis. *partykit: A Modular Toolkit for Recursive Partytioning in R.* Journal of Machine Learning Research, 2015, 16, pp. 3905-3909. [dostęp: 1 marca 2019], <http://jmlr.org/papers/v16/hothorn15a.html>.

[22] Hemant Ishwaran, Udaya Kogalur. *Random Forests for Survival, Regression, and Classification (RF-SRC).* R package version 2.8.0, 2019.

[23] Hemant Ishwaran, Udaya Kogalur. *Random survival forests for R.* R News, 2007, 7(2), pp. 25-31.

[24] Hemant Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone, Michael S. Lauer. *Random survival forests.* Ann. Appl. Statist, 2008, 2(3), pp. 841-860.

[25] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu. *Dokumentacja pakietu lightGBM.* [dostęp: 12 listopada 2018], <https://lightgbm.readthedocs.io/en/latest/index.html>.

[26] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu. *LightGBM: A Highly Efficient Gradient Boosting Decision Tree.* Advances in Neural Information Processing Systems 30, NIPS, 2017, pp.3149-3157.

[27] Guolin Ke. *lightgbm: Light Gradient Boosting Machine.* R package version 2.0.7, 2017. [dostęp: 12 listopada 2018], <https://github.com/Microsoft/LightGBM>.

[28] Jacek Koronacki, Jan Ćwik. *Statystyczne systemy uczące się.* Akademicka Oficyna Wydawnicza EXIT, 2008.

[29] Erin LeDell, Navdeep Gill, Spencer Aiello, Anqi Fu, Arno Candel, Cliff Click, Tom Kraljevic, Tomas Nykodym, Patrick Aboyoun, Michal Kurka and Michal Malohlava. *h2o: R Interface for 'H2O'.* R package version 3.22.1.1, 2019. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=h2o>.

[30] Andy Liaw, Matthew Wiener. *Classification and Regression by randomForest.* R News, 2002, 2(3), pp. 18-22.

[31] Didrik Nielsen. *Tree Boosting With XGBoost.* Praca magisterska, Norwegian University of Science and Technology, Department of Mathematical Sciences, 2016.

[32] Mark O'Connell, Catherine B. Hurley, Katarina Domijan. *Conditional Visualization for Statistical Models: An Introduction to the condvis Package in R.* Journal of Statistical Software, 2017, 81 (5), pp. 1-20. [dostęp: 1 marca 2019], <http://doi.org/10.18637/jss.v081.i05>.

[33] Aleksandra Paluszyńska. *Structure mining and knowledge extraction from random forest with applications to The Cancer Genome Atlas project.* Praca magisterska, Uniwersytet Warszawski, 2017.

[34] Aleksandra Paluszynska, Przemyslaw Biecek . *randomForestExplainer: Explaining and Visualizing Random Forests in Terms of Variable Importance.* R package version 0.9, 2017. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=randomForestExplainer>.

[35] Thomas Lin Pedersen, Michaël Benesty. *lime: Local Interpretable Model-Agnostic Explanations.* R package version 0.4.1, 2018. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=lime>.

[36] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier.* ACM Press, 2016, pp.1135–1144.

[37] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin. *Anchors: High-Precision Model-Agnostic Explanations.* AAAI Conference on Artificial Intelligence, 2018.

[38] Brian Ripley. *tree: Classification and Regression Trees.* R package version 1.0-39, 2018. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=tree>.

[39] Marko Robnik-Sikonja, Petr Savicky. *CORElearn: Classification, Regression and Feature Evaluation.* R package version 1.53.1, 2019. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=CORElearn>.

[40] Rebecca J. Sela, Jeffrey S. Simonoff. *REEMtree: Regression Trees with Random Effects.* R package version 0.90.3, 2011.

[41] Mark Seligman. *Rborist: Extensible, Parallelizable Implementation of the Random Forest Algorithm.* R package version 0.1-8, 2017. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=Rborist>.

[42] Mateusz Staniak, Przemyslaw Biecek. *Explanations of Model Predictions with live and breakDown Packages.* ArXiv e-prints, 1804.01955, 2018. [dostęp: 20 grudnia 2018], <https://arxiv.org/abs/1804.01955>.

[43] Terry Therneau, Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees.* R package version 4.1-13, 2018. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=rpart>.

[44] Jonathan Wand, Gary King, Olivia Lau. *anchors: Statistical analysis of surveys with anchoring vignettes.* R package version 3.0-8, 2016. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=anchors>.

[45] Denis White, Robert B. Gramacy. *maptree: Mapping, pruning, and graphing tree models.* R package version 1.4-7, 2012. [dostęp: 1 marca 2019], <https://CRAN.R-project.org/package=maptree>.

[46] Marvin N. Wright, Andreas Ziegler. *ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R.* Journal of Statistical Software, 2017, 77(1), pp.1-17.