Politechnika Wrocławska

Wydział Matematyki
Kierunek studiów: Matematyka stosowana
Specjalność: –

Praca dyplomowa – inżynierska

WYKRYWANIE OSZUSTW NA KARTACH PŁATNICZYCH Z WYKORZYSTANIEM METOD WRAŻLIWYCH NA KOSZT

Patryk Wielopolski

Słowa kluczowe:
uczenie maszynowe; klasyfikacja wrażliwa na koszt; wykrywanie oszustw na kartach kredytowych

Krótkie streszczenie:
W pracy zaimplementowano eksperyment, który pozwolił porównać klasyczne oraz wrażliwe na koszt metody predykcyjne w problemie detekcji oszustw na kartach płatniczych. W pracy rozważono dwa rodzaje podejść do modelowania wrażliwego na koszt – klasyfikację wrażliwą na koszt oraz trening wrażliwy na koszt. Na podstawie wyników eksperymentów udało się pokazać, że takie podejście do modelowania daje znacznie lepsze rezultaty niż standradowe metody. Ponadto zostało pokazane, że niewykorzysytwany do tej pory w takich problemach algorytm XGBoost w połączeniu z minimalizacją ryzyka bayesowskiego uzyskuje lepsze rezultaty niż wcześniej wykorzysytwane metody.

Opiekun pracy dyplomowej	dr inż. Andrzej Giniewicz	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	ocena	podpis

*Do celów archiwalnych pracę dyplomową zakwalifikowano do:**

a) kategorii A (akta wieczyste)

b) kategorii BE 50 (po 50 latach podlegające ekspertyzie)

** niepotrzebne skreślić*

pieczątka wydziałowa

Wrocław, rok 2020

Wrocław University
of Science and Technology

Faculty of Pure and Applied Mathematics
Field of study: Applied Mathematics
Specialty: –

Engineering Thesis

CREDIT CARD FRAUD DETECTION USING COST SENSITIVE METHODS

Patryk Wielopolski

Keywords:
Machine Learning; Cost Sensitive Classification; Credit Card Fraud Detection

Short summary:
An experiment has been implemented in the work that allowed to compare classical and cost-sensitive methods in credit card fraud detection. The study considers two types of cost-sensitive modeling approaches – cost-sensitive classification and cost-sensitive training. Based on the results of the experiments, it was shown that this modeling approach gives much better results than standard methods. In addition, we have shown that the XGBoost algorithm not used so far in such problems in combination with Bayesian Minimum Risk achieves better results than the previously used methods.

Supervisor	dr inż. Andrzej Giniewicz	.....	.....
	Title/degree/name and surname	grade	signature

*For the purposes of archival thesis qualified to:**

a) category A (perpetual files)

b) category BE 50 (subject to expertise after 50 years)

** delete as appropriate*

stamp of the faculty

Wrocław, 2020

Spis treści

Wstęp

Historia kart płatniczych zaczyna się już w latach osiemdziesiątych XIX wieku, kiedy Edward Bellamy w swojej powieści *Looking Backward* wspomina o możliwości skorzystania z karty przedpłaconej, która umożliwiała dokonanie płatności z wcześniej wpłaconych środków [20]. Niestety pomysł takiej formy płatności nie został pozytywnie przyjęty w tamtych czasach i dopiero w 1914 roku amerykańska firma Western Union umożliwiła korzystanie z kart obciążeniowych, w której bank udziela klientowi odpowiedniego limitu płatności, za które należy zapłacić później. Koncepcja karty bardzo szybko rozpowszechniła się i mimo wybuchu II Wojny Światowej, która wstrzymała jej rozwój, to już w 1950 roku firma Diners Club umożliwiła swoim klientom dokonywania płatności w sklepach, restauracjach lub stacjach benzynowych bez użycia gotówki[1]. Kilka lat później Bank of America wydał pierwszą prawdziwą kartę kredytową, która umożliwiała spłatę zadłużenia w całości lub tylko pewnej wyliczonej kwoty minimalnej. Kolejnym przełomem było wprowadzenie w roku 1972 paska magnetycznego, który umożliwiał korzystanie z numeru PIN i dzięki temu zwiększał bezpieczeństwo klientów [21]. W latach 1973 i 1974 wprowadzono odpowiednio system BASE I oraz BASE II, które były pierwszymi systemami elektronicznymi. Następnie w latach dziewięćdziesiątych pojawiają się technologie, które mają zapewnić zwiększenie bezpieczeństwa oraz zmniejszenie skali oszustw.

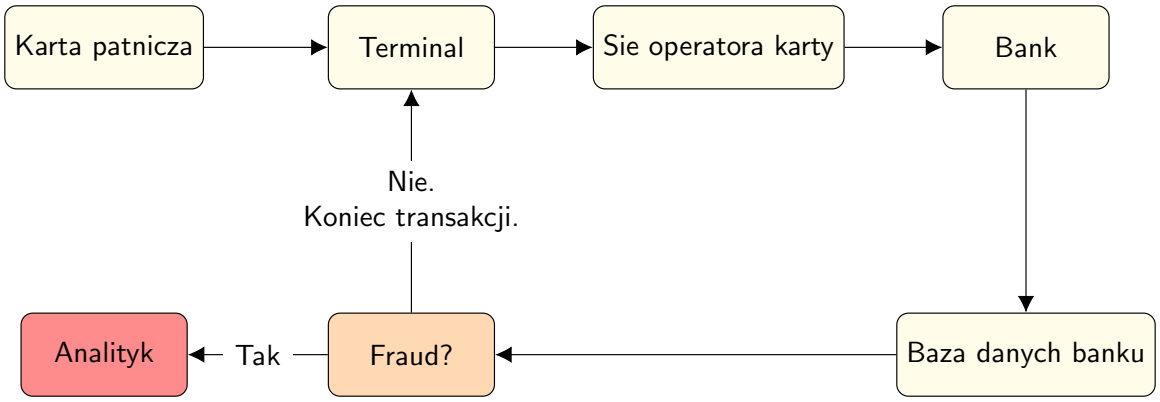
Dalszy rozwój technologiczny umożliwił nie tylko dokonywanie płatności kartą, lecz także wykorzystywanie do tego celu telefonu komórkowego lub zegarka. Ponadto transakcji nie dokonujemy już tylko w sklepach stacjonarnych, lecz także coraz częściej korzystamy ze sklepów internetowych. Wszystkie te czynniki powodują, że dokonywanie zakupów jest jeszcze prostsze, a ilość dokonywanych transakcji z roku na rok rośnie[2]. Niestety ten wzrost spowodował również rozwój sposobów dokonywania przestępstw związanych z kartami płatniczymi, przykładowo przekazanie numeru karty nieznajomym, utrata lub kradzież karty, skopiowania danych karty lub kradzież przesyłki z kartą, które najczęściej kończą się nieautoryzowanymi płatności na koncie.

Banki oraz instytucje finansowe w celu przeciwdziałania tym przestępstwom korzystają z systemów detekcji oszustw. Jest to jeden z elementów całego procesu dokonywania płatności, który jest przedstawiony na rysunku 1 na stronie 2. W momencie przyłożenia karty kredytowej do terminala nawiązywane jest połączenie z operatorem karty (np. Visa, MasterCard), który dalej komunikuje się z bankiem klienta. Następnie w odpowiednich bazach danych sprawdzane są dostępne środki, a system detekcji oszustw sprawdza, czy transakcja nie jest podejrzana. W przypadku wykrycia niezgodności system ją odrzuca i kieruje sprawę do odpowiedniego analityka. W przeciwnym razie płatność jest akceptowana i na terminalu wyświetla się odpowiedni komunikat.

Do niedawna najbardziej popularnym sposobem do wykrywania oszustw był system

[1] Źródło: `https://www.kartyonline.pl/historia-kart-platniczych.php` (dostęp: 27.12.2019)

[2] Źródło: `https://worldpaymentsreport.com/non-cash-payments-volume/` (dostęp: 27.12.2019)



Rysunek 1: Schemat płatności kartą płatniczą. Źródło: Opracowanie własne na podstawie prezentacji A. Bahnsena [1].

oparty na regułach eksperckich [1]. Powstawały one na bazie doświadczenia analityków, którzy podczas swojej długoletniej pracy wielokrotnie sprawdzali transakcje pod kątem oszustw i byli w stanie dostrzec pewne zależności. Przykładami takich reguł mogą być: więcej niż cztery wypłaty z bankomatu w ciągu godziny, więcej niż dwie transakcje w ciągu 5 minut, dwie transakcje w ciągu 5 minut różnymi formami płatności (karta, internet). W przypadku, gdy rozpatrywana transakcja spełnia chociaż jedną z reguł, to jest ona blokowana i użytkownik jest informowany o odrzuceniu transakcji. To podejście jest stosunkowo łatwe do implementacji oraz bardzo proste do interpretacji, lecz niestety ma swoje wady. Przestępcy często wraz z upływem czasu zmieniają swoje sposoby dokonywania oszustw, natomiast systemy złożone z reguł eksperckich nie są w stanie same wykrywać nowych zachowań i dostosowywać się do zmian, lecz wymagają do tego dodatkowej pracy analityków. Tę niedogodność są w stanie rozwiązać modele uczenia maszynowego, które tworzą pewnego rodzaju reguły, które nie są bezpośrednio wprowadzane przez użytkownika, lecz automatycznie wykrywane przez algorytmy na podstawie dostarczonych danych historycznych. Wraz z dynamicznym rozwojem tej dziedziny nauki zaczęły powstawać coraz bardziej wyrafinowane modele, które coraz lepiej radzą sobie z wykrywaniem takich zależności. Obecnie są one już wykorzystywane w produkcyjnych środowiskach instytucji finansowych i wspomagają walkę z przestępcami [12]. Niestety standardowe podejście do modelowania predykcyjnego z wykorzystaniem uczenia maszynowego zakłada równy koszt popełnienia błędu, co nie jest założeniem odpowiadającym rzeczywistości. Stąd powstała motywacja do rozwoju metod, które biorą pod uwagę tę różnicę.

Celem pracy jest implementacja eksperymentu, który pozwoli porównać standardowe oraz wrażliwe na koszt metody predykcyjne w problemie detekcji oszustw na kartach płatniczych. Pierwszy rozdział zawiera wstęp teoretyczny, który wprowadzi odpowiednie pojęcia i modele z dziedziny statystyki matematycznej oraz uczenia maszynowego. Następnie w rozdziale drugim opiszemy przeprowadzone doświadczenie na danych rzeczywistych, w którym zostały wykorzystane modele takie jak regresja logistyczna, drzewa decyzyjne, lasy losowe i algorytm XGBoost wraz z odpowiednimi modyfikacjami oraz metody optymalizacji progu i minimalizacji ryzyka bayesowskiego. Ponadto rozdział zawiera omówienie otrzymanych rezultatów oraz wyciągniętych wniosków. Ostatnia część składa się z podsumowania.

Rozdział 1

Wprowadzenie teoretyczne

Wprowadzenie teoretyczne zawiera przegląd metod z dziedziny statystyki oraz uczenia maszynowego, które wykorzystuje się w zagadnieniach klasyfikacyjnych. W szczególności zostanie opisana macierz pomyłek, macierz kosztu oraz związane z nimi miary skuteczności modeli predykcyjnych, które służą do badania jakości predykcji. Następnie zostaną omówione modele standardowe, które nie biorą pod uwagi różnych kosztów popełniania błędu klasyfikacyjnego. W ramach tej grupy zostanie opisana regresja logistyczna, drzewa decyzyjne, lasy losowe oraz algorytm XGBoost. Na końcu zostanie poruszony temat modeli wrażliwych na koszt, które dzielimy na dwie podkategorie: trening wrażliwy na koszt (*ang. Cost Sensitive Training*) oraz klasyfikacja wrażliwa na koszt (*ang. Cost Sensitive Classification*) [16]. Pierwsza z nich zawiera metody, które w trakcie uczenia modelu zwracają uwagę na koszt błędnej klasyfikacji. Są to rozszerzone wersje standardowych modeli, które w trakcie uczenia się mają zmienioną postać optymalizowanej funkcji. W niniejszej pracy będziemy omawiać regresję logistyczną wrażliwą na koszt oraz drzewa decyzyjne wrażliwe na koszt. Druga podkategoria składa się z metod, które wykorzystują estymowane prawdopodobieństwo ze standardowych modeli i na jego podstawie tworzą dodatkowy model, który bierze pod uwagę koszt niepoprawnej klasyfikacji danego przypadku. Z tej grupy zostanie omówiona optymalizacja progu oraz minimalizacja ryzyka bayesowskiego.

W pracy wykorzysytwane są następujące oznaczenia:

- $\mathcal{S} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ – zbiór wszystkich N obserwacji,
- $\boldsymbol{x}_i = \left(x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(j)}\right)$ – i-ty wektor obserwacji składający się z j atrybutów,
- $x_i^{(k)}$ – k-ty atrybut i-tego wektora obserwacji,
- $\boldsymbol{y} = (y_1, y_2, \ldots, y_N)$ – wektor prawdziwych stanów klasyfikacji,
- $y_i \in \{0, 1\}$ – prawdziwy stan klasyfikacji dla i-tej obserwacji,
- $\boldsymbol{p} = (p_1, p_2, \ldots, p_N)$ – wektor estymowanych wartość prawdopodobieństwa,
- $p_i = P(y_i = 1|\mathcal{S})$ – estymowana wartość prawdopodobieństwa dla i-tej obserwacji,
- $\boldsymbol{c} = (c_1, c_2, \ldots, c_N)$ – wektor przewidywanych klas,
- $c_i \in \{0, 1\}$ – przewidywana klasa dla i-tej obserwacji.

Wartość $y_i = 0$ oznacza wartość niesprzyjającą a $y_i = 1$ sprzyjającą. Ponadto $c_i = 0$ oznacza klasyfikację negatywną, natomiast $c_i = 1$ pozytywną.

1.1 Miary skuteczności modeli

1.1.1 Macierz pomyłek

Jedną z metod wykorzystywanych do oceny skuteczności modeli w zadaniach klasyfikacyjnych jest macierz pomyłek. Prawidłowe stany zestawiamy z klasami nadanymi przez model, a następnie sprawdzamy, czy poprawnie zaklasyfikowaliśmy poszczególne obserwacje. W przypadku klasyfikacji binarnej popełniamy dwa rodzaje błędów, a wszystkie możliwe wyniki prezentują się w następujący sposób:

– Prawdziwie pozytywny wynik klasyfikacji (TP, z angielskiego *True Positive*) – obserwacja miała stan sprzyjający i zaklasyfikowaliśmy ją jako pozytywną;

– Fałszywie pozytywny wynik klasyfikacji (FP, z angielskiego *False Positive*) – obserwacja miała stan niesprzyjający i zaklasyfikowaliśmy ją jako pozytywną. W nomenklaturze statystycznej nazywamy ją również błędem pierwszego rodzaju;

– Fałszywie negatywny wynik klasyfikacji (FN, z angielskiego *False Negative*) – obserwacja miała stan sprzyjający i zaklasyfikowaliśmy ją jako negatywną. W nomenklaturze statystycznej nazywamy ją również błędem drugiego rodzaju;

– Prawdziwie negatywny wynik klasyfikacji (TN, z angielskiego *True Negative*) – obserwacja miała stan niesprzyjający i zaklasyfikowaliśmy ją jako negatywną.

Liczbę obserwacji należących do poszczególnych kategorii możemy reprezentować tak jak w tabeli 1.1, gdzie

- $\text{TP} = \#\{i : i \in \{1, \ldots, N\}, y_i = c_i = 1\}$,
- $\text{FP} = \#\{i : i \in \{1, \ldots, N\}, y_i = 0, c_i = 1\}$,
- $\text{FN} = \#\{i : i \in \{1, \ldots, N\}, y_i = 1, c_i = 0\}$,
- $\text{TN} = \#\{i : i \in \{1, \ldots, N\}, y_i = c_i = 0\}$.

	Stan sprzyjający $y_i = 1$	Stan niesprzyjający $y_i = 0$
Predykcja pozytywna $c_i = 1$	TP	FP
Predykcja negatywna $c_i = 0$	FN	TN

Tabela 1.1: Macierz pomyłek.

1.1.2 Miary skuteczności niewrażliwe na koszt

Na podstawie macierzy pomyłek (tabela 1.1) możemy zdefiniować następujące miary skuteczności modeli, które nie biorą pod uwagę kosztu popełnienia błędu. Przytoczone definicje pochodzą z pracy D. Powersa [17].

Precyzja

Precyzja (*ang. Precision*) to stosunek poprawnie zaklasyfikowanych pozytywnych przypadków do wszystkich obserwacji zaklasyfikowanych jako pozytywne. Mierzy ona, jaki procent próbek prawidłowo zaklasyfikowaliśmy jako pozytywne. Warto zauważyć, że nie bierze ona pod uwagę klasyfikacji negatywnych. Jest ona bardzo często wykorzystywana podczas rozwiązywania większości problemów z zakresu uczenia maszynowego, ponieważ najczęściej interesuje nas, aby nasze typowania były możliwie najbardziej skuteczne. Z tego też powodu precyzja jest często nazywana skutecznością klasyfikacji prawdziwie pozytywnych (*ang. True Positive Accuracy*). Zapisana wzorem przyjmuje postać

$$\text{Precyzja} = \frac{TP}{TP + FP}.$$

Rozpatrzmy jeszcze przykład z medycyny. Załóżmy, że mamy test diagnostyczny, który klasyfikuje wszystkich pacjentów jako chorych. W tym przypadku czułość przyjmuje wartość 1, swoistość wartość 0, a precyzja jest równa frakcji przypadków pozytywnych w danych. Odwrotnie, załóżmy teraz, że nasz test oznacza wszystkich pacjentów jako zdrowych. W tym przypadku czułość jest równa 0, swoistość 1, natomiast precyzja jest niezdefiniowana (symbol nieoznaczony – $\frac{0}{0}$). Ten przykład pokazuje, że precyzja jest zależna od zbalansowania klas.

Czułość

Czułość (*ang. Recall* lub *ang. Sensitivity*) to stosunek prawdziwie pozytywnie zaklasyfikowanych przypadków do wszystkich sprzyjających obserwacji. Innymi słowy, jest to miara, która mówi nam o procencie znalezionych przez model przypadków sprzyjających. Podobnie jak precyzja skupia się jedynie na poprawnie zaklasyfikowanych obserwacjach pozytywnych. Jest ona szczególnie ważna w zastosowaniach medycznych, gdzie naszym głównym celem jest znalezienie wszystkich chorych osób. Używana jest także do wyznaczania krzywej ROC. Czułość jest wyrażona wzorem

$$\text{Czułość} = \frac{TP}{TP + FN}.$$

Swoistość

Swoistość (*ang. Specificity*) to stosunek prawdziwie negatywnie zaklasyfikowanych przypadków do wszystkich niesprzyjających obserwacji. Inaczej nazywana też odwrotną czułością, gdyż podobnie do czułości mierzy ona odsetek znalezionych wszystkich obserwacji, lecz dla negatywnej klasy. Miara ta jest także używana do wyrysowania krzywej ROC, a dokładniej wartość $1 -$ Swoistość, która odpowiada stosunkowi fałszywie pozytywnych klasyfikacji (ang. *false positive ratio*). Swoistość wyraża się następującym wzorem

$$\text{Swoistość} = \frac{TN}{TN + FP}.$$

Dokładność

Dokładność (*ang. Accuracy*) mierzy stosunek poprawnie zaklasyfikowanych przypadków do ilości wszystkich obserwacji. Bierze ona pod uwagę zarówno obserwacje pozytywne, jak i negatywne. Z tego powodu bardzo dobrze radzi sobie w sytuacji, gdy interesują

nas obserwacje z obu klas, natomiast dużo gorzej, gdy istotna jest tylko jedna klasa. W szczególności nie nadaje się ona do problemów z dużą dysproporcją klas, która pojawia się w przypadku detekcji oszustw. Dokładność zdefiniowana jest następującym wzorem

$$\text{Dokładność} = \frac{TP + TN}{TP + FP + FN + TN},$$

który możemy zapisać również jako (wyprowadzenie w dodatku A)

$$\text{Dokładność} = \alpha \cdot \text{Czułość} + (1 - \alpha) \cdot \text{Swoistość}, \tag{1.1}$$

gdzie α to odsetek przypadków sprzyjających w danych.

Rozpatrzmy pewien przykład. Załóżmy, że mamy dwóch naukowców, którzy posługują się tą samą metodą w badaniach, lecz mają oni różne proporcje stanów w populacji. W tym przypadku otrzymają oni różną dokładność, mimo tego, że wyniki nie powinny się różnić z uwagi na korzystanie z tej samej metody. W celu uniknięcia takiej sytuacji możemy wykorzystać miarę zbalansowanej skuteczność (*BACC, ang. balanced accuracy*). Dana jest ona wzorem 1.1, gdzie za parametr α przyjmuje się wartość $\frac{1}{2}$. W przypadku, gdy mamy równe proporcje klas, to sprowadza się ona do dokładności, natomiast gdy standardowa dokładność jest wysoka tylko z uwagi na niezbalansowaną liczność próbek w klasach, to wartość zbalansowanej skuteczności spada [8]. Jest ona również ważna, ponieważ maksymalizacja tej miary odpowiada standardowemu szukaniu optymalnego punktu na krzywej ROC według metryki taksówkowej.

Krzywa ROC

Krzywa ROC to wykres diagnostyczny modelu predykcyjnego pozwalający na zilustrowanie jakości predykcyjnej klasyfikatora. Składa się ona z wyrysowanym na osi X stosunku fałszywie pozytywnych klasyfikacji oraz czułości na osi Y [17]. Miary są narysowane dla różnych wartości progu decyzyjnego, który prawdopodobieństwa zwracane przez model zamienia na decyzje binarne. Perfekcyjny klasyfikator jest w stanie osiągnąć lewy górny róg wykresu, czyli posiadać precyzję na poziomie 100% oraz stosunek fałszywie pozytywnych klasyfikacji na poziomie 0%. Inaczej, taki model ma czułość oraz precyzję na poziomie 100%, a co za tym idzie 100% dokładność, a to oznacza, że nigdy się nie myli. Analogicznie najgorszy klasyfikator jest w stanie osiągnąć prawy dolny róg wykresu. Modelem losowy będzie punkt na prostej z punktu $(0, 0)$ do punktu $(1, 1)$, czyli każda dowolna klasyfikacja z rozkładu $B(1, p)$ niezależnego od X jest w punkcie (p, p). Ponadto warto zauważyć, że klasyfikatory z przykładu w sekcji dotyczącej precyzji (wszyscy chorzy i wszyscy zdrowi) też tutaj są – na końcach tego odcinka w punkcie $(1, 1)$ oraz $(0, 0)$ odpowiednio.

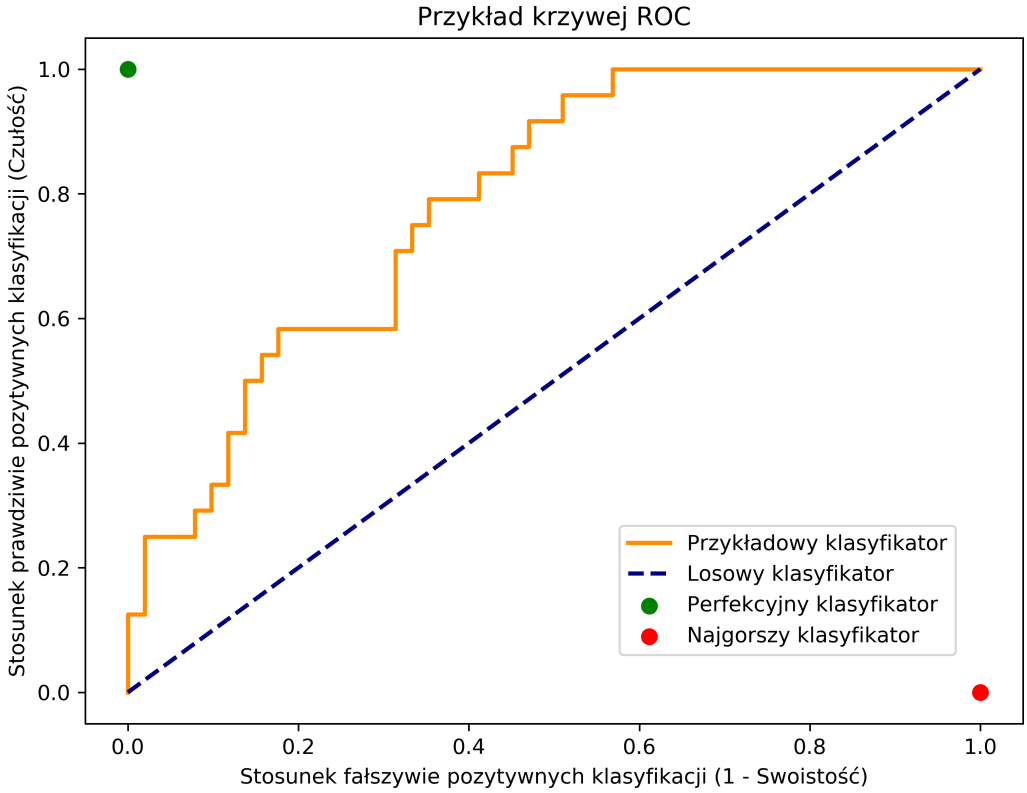
Przykładowy wykres znajduje się na rysunku 1.1 na stronie 7. Kolorem czerwonym oznaczony jest najgorszy możliwy klasyfikator, zielonym perfekcyjny, niebieskim losowy, natomiast żółtym przykładowy.

Celem tworzenia oraz ulepszania kolejnych modeli predykcyjnych jest zbliżanie się w wynikach do lewego górnego rogu wykresu. Odległość danego punktu reprezentującego konkretny model możemy definiować w różnych metrykach. Przykładowo w metryce taksówkowej (metryka L_1) będzie to

$$1 - \text{Swoistość} + 1 - \text{Czułość}.$$

Dokonując kolejne przekształcenia otrzymujemy

$$\begin{aligned} 1 - \text{Swoistość} + 1 - \text{Czułość} &= 2 - (\text{Czułość} + \text{Swoistość}) = \\ &= 2 \cdot (1 - (\text{Czułość} + \text{Swoistość})/2) = 2 \cdot (1 - \text{BACC}). \end{aligned}$$



Rysunek 1.1: Przykładowy wykres charakterystyki pracy odbiornika. Na poziomej osi przedstawiony jest stosunek fałszywie pozytywnych klasyfikacji, natomiast na pionowej czułość. Kolorem czerwonym, zielonym, niebieskim oraz żółtym jest odpowiednio oznaczony najgorszy, najlepszy, losowy oraz przykładowy klasyfikator. Źródło: Opracowanie własne.

Oznacza to, że im większa zbalansowana skuteczność danego modelu, tym bliżej znajduje się model na krzywej ROC lewego górnego rogu. W podobny sposób możemy także rozważać inne metryki. Przykładowo w metryce euklidesowej (metryka L_2) będziemy chcieli minimalizować następujące wyrażenie

$$\sqrt{(1-\text{Swoistość})^2 + (1-\text{Czułość})^2},$$

bądź ogólniej w metryce L_p

$$\Big((1-\text{Swoistość})^p + (1-\text{Czułość})^p\Big)^{\frac{1}{p}}.$$

F Score

F_1 Score to miara łącząca w sobie precyzję oraz czułość za pomocą średniej harmonicznej. Z uwagi na wykorzystanie tych dwóch wartości nie bierze ona pod uwagę prawidłowo zaklasyfikowanych obserwacji negatywnych. Wykorzystanie średniej harmonicznej powoduje, że kładziemy równy nacisk na obie użyte miary. Standardowy wzór wygląda następująco

$$F_1 = \left(\frac{2}{\text{Precyzja}^{-1} + \text{Czułość}^{-1}}\right) = 2 \cdot \frac{\text{Precyzja} \cdot \text{Czułość}}{\text{Precyzja} + \text{Czułość}}.$$

W przypadku, gdy interesuje nas bardziej precyzja lub czułość, to definiujemy bardziej ogólną formułę na F_β Score:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precyzja} \cdot \text{Czułość}}{(\beta^2 \cdot \text{Czułość}) + \text{Precyzja}},$$

gdzie parametr $\beta > 0$ odpowiada na pytanie, ile razy bardziej czułość jest ważna niż precyzja. Najczęściej wykorzystuje się parametr $\beta = 2$ lub $\beta = \frac{1}{2}$, który odpowiednio zwiększa nacisk na czułość lub zmniejsza wpływ fałszywie negatywnie zaklasyfikowanych obserwacji, skupiając się bardziej na precyzji.

1.1.3 Macierz kosztu

W celu wprowadzenia potrzebnych miar skuteczności wrażliwych na koszt, musimy zdefiniować wektor macierzy kosztu

$$\boldsymbol{C} = (C_1, C_2, \ldots, C_N)$$

gdzie $C^{(i)} = [C^{(i)}_{j,k}]^1_{j,k=0}$ oznacza macierz kosztu dla i-tej obserwacji [5, 14]. Ma ona postać analogiczną do macierzy pomyłek, lecz zamiast ilości obserwacji w odpowiednich komórkach wpisujemy koszt pomyłki dla i-tej obserwacji. Warto zaznaczyć, że koszt niekoniecznie musi być kwota pieniężną, lecz także możemy rozważać ilość poświęconego czasu, bądź inną dowolną mierzalną jednostkę. W szczególności dla przypadku klasyfikacji binarnej wyróżniamy cztery możliwe wartości:

– $C^{(i)}_{1,1}$ – koszt prawdziwie pozytywnego zaklasyfikowania i-tej obserwacji,

– $C^{(i)}_{1,0}$ – koszt fałszywie pozytywnego zaklasyfikowania i-tej obserwacji,

– $C^{(i)}_{0,1}$ – koszt fałszywie negatywnego zaklasyfikowania i-tej obserwacji,

– $C^{(i)}_{0,0}$ – koszt prawdziwie negatywnego zaklasyfikowania i-tej obserwacji.

W zależności od praktycznego zastosowania wartości w macierzy kosztu przyjmują różne wielkości. Poniżej przedstawiamy kilka intuicji na podstawie macierzy kosztu zdefiniowanych dla różnych eksperymentów w pracy A. Bahnsena [4].

- W przypadku, gdy predykcja klasy pozytywnej wiąże się z podjęciem jakiejś akcji, np. sprawdzenie transakcji kartą kredytową, wysłanie oferty reklamowej, to koszt pozytywnej predykcji jest dodatni, to znaczy $C^{(i)}_{1,1}$, $C^{(i)}_{1,0} > 0$. Ponadto najczęściej te koszty są równe i przyjmują stałą wartość dla wszystkich przypadków, to znaczy $C^{(i)}_{1,1} = C^{(i)}_{1,0} = \text{const.}$, ponieważ podejmowana zawsze jest ta sama akcja dla wszystkich obserwacji. Dodatkowo koszt pozytywnego zaklasyfikowania dla każdej obserwacji negatywnej jest zerowy, to znaczy $C^{(i)}_{0,0} = 0$, ponieważ słusznie nie została podjęta żadna akcja. Zatem ostatecznie jedyną wartością, która różni się między przypadkami, jest koszt fałszywie negatywnej klasyfikacji i on również jest większy od zera.

- W przypadku, gdy predykcja dokonywana jest przez zautomatyzowany system, to koszt prawidłowej klasyfikacji jest równy zero, to znaczy $C^{(i)}_{1,1} = C^{(i)}_{0,0} = 0$. Jest to spowodowane faktem, że maszynie nie musimy płacić pensji, a koszt czasowy i sprzętowy jest pomijalny. Natomiast koszty pomyłek są inne i zazwyczaj różnią się

między obserwacjami. Biorąc jako przykład ryzyko kredytowe, możemy zauważyć, że koszty mogą być zarówno dodatnie, jak i ujemne. W tym konkretnym przypadku, gdy stwierdzimy, że dana osoba spłaci kredyt, natomiast w rzeczywistości nie zrobi tego, to będziemy stratni pewną sumę pieniędzy, co da nam dodatni koszt. Z drugiej strony, gdy uznamy, że dana osoba nie spłaci kredytu, a w rzeczywistości wywiązałaby się ze zobowiązań, to utracimy pewien zarobek, zatem koszt będzie ujemny.

Koncepcyjnie koszt popełnienia błędu dla każdej obserwacji powinien być większy niż koszt poprawnej klasyfikacji, to znaczy

$$C_{1,0}^{(i)} > C_{1,1}^{(i)} \text{ i } C_{0,1}^{(i)} > C_{0,0}^{(i)} \quad \forall_{i\in\{1,2,\ldots,N\}}.$$

Ilustracja takiej macierzy kosztu znajduje się w tabeli 1.2.

	Stan pozytywny $y_i = 1$	Stan negatywny $y_i = 0$
Predykcja pozytywna $c_i = 1$	$C_{1,1}^{(i)}$	$C_{1,0}^{(i)}$
Predykcja negatywna $c_i = 0$	$C_{0,1}^{(i)}$	$C_{0,0}^{(i)}$

Tabela 1.2: Macierz kosztu dla i-tej obserwacji.

1.1.4 Miary skuteczności modeli wrażliwych na koszt

Motywacją do powstania miar skuteczności wrażliwych na koszt jest potrzeba ewaluacji modeli, która miarodajnie odda złożoność rozwiązywanego problemu [2]. Przykładowo w przypadku detekcji oszustw będzie w stanie odpowiedzieć na pytanie dotyczące zaoszczędzonej kwoty dzięki modelowi predykcyjnemu. Kolejny powodem jest fakt, że podstawowe metryki nie uwzględniają różnicy w kosztach pomyłki dla fałszywie pozytywnych oraz fałszywie negatywnych klasyfikacji, traktując oba błędy jednakowo. Ponadto warto zauważyć, że omawiane problemy wymagają uwzględnienia nie tylko różnych kosztów dla każdego rodzaj pomyłki, ale także każdej konkretnej obserwacji. Bazując na wprowadzonym w sekcji 1.1.3 wektorze macierzy kosztu C, można zdefiniować następujące miary.

Koszt całkowity

Koszt całkowity (TC, z angielskiego *ang. Total Cost*) to suma kosztów poniesionych dla danego zestawu predykcji $\boldsymbol{c}$, który zostały wygenerowany przez pewien model predykcyjny. Definiowany jest wzorem

$$\mathrm{TC}(\boldsymbol{y}, \boldsymbol{c}, \boldsymbol{C}) = \sum_{i=1}^{N} C_{c_i,y_i}^{(i)}. \tag{1.2}$$

Oszczędności

Oszczędności (*ang. Savings*) to miara, którą intuicyjnie możemy rozumieć jako procentową wartość, o ile testowany model jest lepszy od naiwnego klasyfikatora. Przy czym naiwnym klasyfikatorem nazywamy model, który zwraca same predykcje pozytywne lub

same predykcje negatywne, w zależności od tego, co bardziej się opłaca dla danego zestawu danych.

$$\text{Oszczędności}(\boldsymbol{y}, \boldsymbol{c}, \boldsymbol{C}) = \frac{\text{Koszt bazowy}(\boldsymbol{y}, \boldsymbol{C}) - \text{TC}(\boldsymbol{y}, \boldsymbol{c}, \boldsymbol{C})}{\text{Koszt bazowy}(\boldsymbol{y}, \boldsymbol{C})}, \tag{1.3}$$

gdzie:

- koszt bazowy$(\boldsymbol{y}, \boldsymbol{C}) = \min\{\text{TC}(\boldsymbol{y}, \boldsymbol{c}_0, \boldsymbol{C}), \text{TC}(\boldsymbol{y}, \boldsymbol{c}_1, \boldsymbol{C})\} \neq 0$ – koszt poniesiony w przypadku używania naiwnego klasyfikatora,
- $\boldsymbol{c}_0 = (0, 0, \ldots, 0)$ – N-elementowy wektor predykcji równych 0,
- $\boldsymbol{c}_1 = (1, 1, \ldots, 1)$ – N-elementowy wektor predykcji równych 1.

1.2 Standardowe modele

1.2.1 Regresja logistyczna

Regresja logistyczna to model statystyczny, którego celem jest modelowanie prawdopodobieństwa, że zmienna losowa Y przyjmie wartość 0 lub 1, na podstawie danych empirycznych. Rozważmy następującą definicję modelu regresji logistycznej parametryzowaną wektorem współczynników $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$ daną w następujący sposób

$$h_\theta(\boldsymbol{x_i}) = \frac{1}{1 + e^{-\boldsymbol{x}_i \theta^T}} = P(Y = 1 | X = \boldsymbol{x}_i; \theta). \tag{1.4}$$

Jednocześnie zauważmy, że

$$P(Y = 0 | X = \boldsymbol{x}_i; \theta) = 1 - P(Y = 1 | X = \boldsymbol{x}_i; \theta) = 1 - h_\theta(\boldsymbol{x_i})$$

oraz korzystając z faktu, że $y_i \in \{0, 1\}$, możemy zapisać

$$P(Y = y_i | X = \boldsymbol{x}_i; \theta) = h_\theta(\boldsymbol{x_i})^{y_i} (1 - h_\theta(\boldsymbol{x_i}))^{(1-y_i)}.$$

Następnie wyznaczamy funkcję wiarogodności, aby wyestymować parametry θ.

$$L(\theta | \mathcal{S}, \boldsymbol{y}) = \prod_{i=1}^{N} P(Y = y_i | X = \boldsymbol{x}_i; \theta) = \prod_{i=1}^{N} h_\theta(\boldsymbol{x_i})^{y_i} (1 - h_\theta(\boldsymbol{x_i}))^{(1-y_i)}.$$

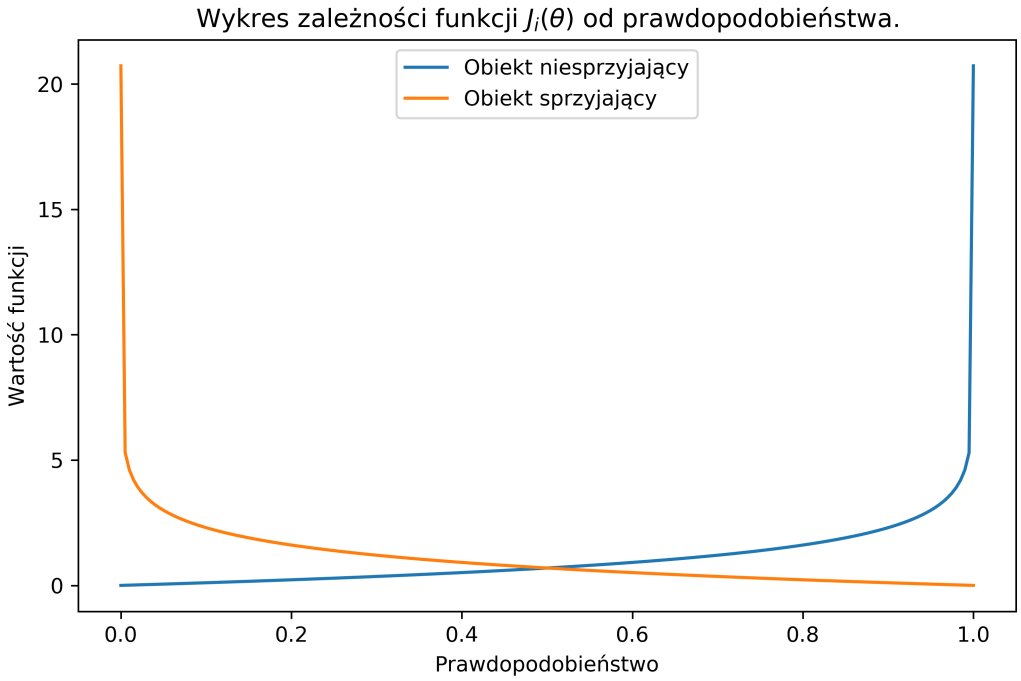
W celu znalezienia maksimum funkcji wiarogodności logarytmujemy funkcję $L(\theta | \mathcal{S}, \boldsymbol{y})$ i otrzymujemy funkcję straty $J(\theta)$

$$J(\theta) = \log L(\theta | \mathcal{S}, \boldsymbol{y}) = -\sum_{i=1}^{N} J_i(\theta),$$

gdzie $J_i(\theta)$ nazywana jest entropią krzyżową i przyjmuje postać

$$J_i(\theta) = -\Big(y_i \log(h_\theta(\boldsymbol{x_i})) + (1 - y_i) \log(1 - h_\theta(\boldsymbol{x_i}))\Big). \tag{1.5}$$

Niestety nie jest możliwe znalezienie bezpośredniego wzoru na współczynniki, które maksymalizują funkcję straty $J(\theta)$, zatem wykorzystuje się w tym celu algorytmy optymalizacji numerycznej, np. IRLS (*ang. Iteratively Reweighted Least Squares*). Warto zaznaczyć, że



Rysunek 1.2: Wykres zależności funkcji $J_i(\theta)$ od prawdopodobieństwa predykcji danej obserwacji. Kolor niebieski przedstawia wykres dla próbki o prawdziwej klasie pozytywnej, natomiast kolor pomarańczowy dla negatywnej. Źródło: Opracowanie własne.

w literaturze najczęściej funkcja straty $J(\theta)$ podawana jest bez znaku minus, a algorytm szuka odpowiednio minimum funkcji.

Zauważmy, że funkcja straty składa się z funkcji $J_i(\theta)$ dla poszczególnych obserwacji. W celu lepszego zrozumienia całości przyjrzymy się bliżej własnościom funkcji $J_i(\theta)$.

Wykres 1.2 przedstawia przyjmowane wartości funkcji $J_i(\theta)$ w zależności od estymowanego prawdopodobieństwa $h_\theta(\boldsymbol{x_i})$. Kolorem niebieskim jest oznaczony przebieg funkcji dla obiektu niesprzyjającego, a pomarańczowym dla sprzyjającego. Możemy zauważyć, że dla obu rodzajów obiektów, funkcja $J_i(\theta)$ jest odbiciem funkcji dla przeciwnego obiektu względem prostej pionowej dla prawdopodobieństwa równego 0.5 . Oznacza to, że ten klasyfikator przykłada taką samą wagę dla obu rodzajów błędów. Ponadto rozważając następujące granice dla $y_i = 0$

$$\lim_{h_\theta(\boldsymbol{x_i})\to 0^+} -\log(1 - h_\theta(\boldsymbol{x_i})) = 0,$$

$$\lim_{h_\theta(\boldsymbol{x_i})\to 1^-} -\log(1 - h_\theta(\boldsymbol{x_i})) = \infty$$

oraz dla $y_i = 1$

$$\lim_{h_\theta(\boldsymbol{x_i})\to 0^+} -\log(h_\theta(\boldsymbol{x_i})) = \infty,$$

$$\lim_{h_\theta(\boldsymbol{x_i})\to 1^-} -\log(h_\theta(\boldsymbol{x_i})) = 0$$

otrzymujemy, że koszt klasyfikacji dla pojedynczej obserwacji przyjmuje w przybliżeniu następujące wartości

$$J_i(\theta) \approx \begin{cases} 0, & \text{jeżeli } h_\theta(\boldsymbol{x_i}) \approx y_i, \\ \infty, & \text{jeżeli } 1 - h_\theta(\boldsymbol{x_i}) \approx y_i. \end{cases}$$

To znaczy, że w przypadku prawidłowego zaklasyfikowania danej obserwacji funkcja przyjmuje wartość bliską zeru, natomiast w przypadku pomyłki nieskończoności. Stąd możemy wywnioskować, że koszty popełnienia bądź niepopełnienia błędu są następujące:

$$C_{1,1}^{(i)} = C_{0,0}^{(i)} \approx 0,$$

$$C_{1,0}^{(i)} = C_{0,1}^{(i)} \approx \infty.$$

Jest to wynik, który nie oddaje charakteru nierówności kosztu popełniania różnych błędów, stąd powstaje motywacja, aby zmodyfikować podaną funkcję i stworzyć regresję logistyczną, która będzie w stanie modyfikować koszty poszczególnych klasyfikacji. Przykład takiego modelu jest omówiony w sekcji 1.4.1.

1.2.2 Drzewo decyzyjne

Drzewo decyzyjne to nieparametryczny model uczenia nadzorowanego, który jest wykorzystywany do regresji oraz klasyfikacji. Jego celem jest nauczenie się na podstawie dostępnych danych prostych reguł decyzyjnych. Do jego największych zalet należy łatwość interpretacji otrzymanych wyników oraz niewielka ilość niezbędnych przygotowań danych, ponieważ między innymi nie wymagają one normalizacji, skalowania lub transformacji zmiennych kategorycznych.

Poniżej opiszemy trzy rodzaje najpopularniejszych algorytmów, które są odpowiedzialne za proces uczenia drzew.

- ID3 (Iterative Dichotomiser 3) – algorytm pozwala na tworzenie drzew decyzyjnych, które w poszczególnych węzłach mogą mieć więcej niż dwa podziały. Działa jedynie dla zmiennych kategorycznych. Drzewo rośnie do maksymalnego rozmiaru, a następnie jest przycinane;

- C4.5 – następca ID3, który znosi restrykcję używania tylko kategorycznych atrybutów poprzez dyskretyzację zmiennych ciągłych. Nauczone drzewo zamieniane jest w zestaw reguł, które są szeregowane względem poprawy skuteczności. Przycinanie w tym wypadku polega na usuwanie reguł, które nie poprawiają wyniku modelu;

- CART (z angielskiego *Classification and Regression Tree*) – bardzo podobny algorytm do C4.5, który umożliwia tworzenie drzew regresyjnych (w poprzednich algorytmach mamy do czynienia jedynie z zadaniami klasyfikacyjnymi) oraz nie tworzy zestawu reguł. Konstruuje on binarne drzewa, które do podziału używa atrybutu oraz progu odcięcia maksymalizującego przyrost informacji.

Warto zaznaczyć, że implementacja w bibliotece *sklearn* w Pythonie wykorzystuje zmodyfikowaną wersję algorytmu CART, który nie obsługuje zmiennych kategorycznych. Poniżej prezentujemy teorię matematyczną stojącą za implementacją tego algorytmu [9].

Niech $\mathcal{Q}$ będzie zbiorem obserwacji dla danego węzła m. Ponadto niech podział będzie reprezentowany przez parę $\theta = (j, t_m)$, gdzie j odpowiada j-temu atrybutowi wektora $\boldsymbol{x}_i$, natomiast t_m to warunek podziału danych na podzbiory $\mathcal{Q}_l$ oraz $\mathcal{Q}_r$, gdzie

$$\mathcal{Q}_l(\theta) = \{\boldsymbol{x}_i : \boldsymbol{x}_i \in \mathcal{Q} \wedge x_i^{(j)} \leq t_m\},$$

$$\mathcal{Q}_r(\theta) = \mathcal{Q} \setminus \mathcal{Q}_l(\theta).$$

Czystość danego węzła jest wyznaczana w następujący sposób

$$G(\mathcal{Q}, \theta) = \frac{|\mathcal{Q}^l(\theta)|}{|\mathcal{Q}|} I(\mathcal{Q}_l(\theta)) + \frac{|\mathcal{Q}^r(\theta)|}{|\mathcal{Q}|} I(\mathcal{Q}_r(\theta)),$$

gdzie $I(\cdot)$ może być dowolną funkcją miary zanieczyszczenia. Ostatecznie w danym węźle wybierany jest ten podział, który minimalizuje funkcję czystości

$$\theta^* = \arg\min_{\theta} G(\mathcal{Q}, \theta).$$

Dalej następuje rekurencyjny podział na podzbiory wyznaczonego $\mathcal{Q}_l(\theta^*)$ i niezależnie $\mathcal{Q}_r(\theta^*)$. Algorytm działa do osiągnięcia maksymalnej głębokości drzewa. W zależności od implementacji oraz preferencji użytkownika po zakończeniu działania algorytmu wykonuje się przycinanie drzewa. Jest to metoda, która w ogólności polega na usuwaniu nadmiarowych węzłów, które mogą powodować zbyt dokładne dopasowanie do danych. Ponadto ostatnie podzbiory, które pozostały na końcu drzewa, nazywamy liśćmi. W momencie predykcji wykorzystujemy utworzone podziały, aby dla zadanej obserwacji znaleźć odpowiedni liść. Następnie model zwraca prawdopodobieństwo należenia do poszczególnych klas, które jest równe procentowi próbek z danej klasy w danych treningowych w znalezionym liściu.

W zależności od rodzaju problemu, wykorzystujemy różne miary zanieczyszczenia [9]. Dla zadań klasyfikacyjnych są to

- Misclassification: $I_m(\mathcal{Q}) = 1 - \max(p(\mathcal{Q}), 1 - p(\mathcal{Q}))$,
- Entropy: $I_e(\mathcal{Q}) = -p(\mathcal{Q}) \log_2 p(\mathcal{Q}) - (1 - p(\mathcal{Q})) \log_2(1 - p(\mathcal{Q}))$,
- Gini: $I_g(\mathcal{Q}) = 2p(\mathcal{Q})(1 - p(\mathcal{Q}))$,

a dla zadań regresyjnych

- $I_{mse}(\mathcal{Q}) = \frac{1}{|\mathcal{Q}|} \sum_{i \in \mathcal{Q}} (y_i - q(\mathcal{Q}))^2$,
- $I_{mae}(\mathcal{Q}) = \frac{1}{|\mathcal{Q}|} \sum_{i \in \mathcal{Q}} |y_i - q(\mathcal{Q})|$,

gdzie:

$$p(\mathcal{Q}) = \frac{|\{\boldsymbol{x}_i : \boldsymbol{x}_i \in \mathcal{Q} \wedge y_i = 1\}|}{|\mathcal{Q}|},$$

$$q(\mathcal{Q}) = \frac{\sum_{i \in \mathcal{Q}} y_i}{|\mathcal{Q}|}.$$

1.2.3 Las losowy

Las losowy oraz algorytm XGBoost są przedstawicielami szerokiej klasy modeli typu *ensemble*. Głównym celem tej metodologii jest wprowadzenie losowych perturbacji do zbioru treningowego, w celu utworzenia różnych klasyfikatorów bazowych, których wspólne predykcje będą lepsze niż każdego z modeli bazowych osobno. Ponadto istnieją trzy główne powody, dlaczego te metody działają znacznie lepiej niż pojedyncze modele:

- Statystyczny – w przypadku, gdy mamy do czynienia z małym zbiorem danych, istnieje możliwość stworzenia kilku bardzo dobryh klasyfikatorów, które na zbiorze treningowym mają taką samą skuteczność. W przypadku dodatkowego zbioru testowego istnieje szansa wybrania nieoptymalnego;

- Obliczeniowy – większość algorytmów polega na optymalizacji numerycznej, która może zatrzymać się w minimum lokalnym. Dzięki zastosowaniu kilku modeli uzyskujemy szansę przeszukania większej części przestrzeni parametrów modeli;

- Reprezentacyjny – często funkcja f reprezentująca prawdziwy model jest bardzo skomplikowana i niemożliwe jest przedstawienie jej za pomocą jednego modelu. Wykorzystując złożenie kilku modeli bazowych, mamy szansę lepiej przybliżać prawdziwą funkcję f.

W celu utworzenia T klasyfikatorów bazowych, należy ze zbioru treningowego $\mathcal{S}$ wybrać S_j dla $j = 1, 2, \ldots, T$ losowych podzbiorów obserwacji, które wykorzystamy do uczenia. W tym celu wykorzystuje się następujące procedury losowania.

- Wklejanie (*ang. Pasting*) – Polega na niezależnym losowaniu $K < N$ próbek ze zbioru $\mathcal{S}$ bez powtórzeń. Najczęściej wykorzystywana jest na bardzo dużych zbiorów danych, gdzie zasoby obliczeniowe są ograniczone i nie jest możliwe wczytanie wszystkich danych do pamięci komputera. Ponadto dzięki możliwości wykorzystania obliczeń równoległych skraca się czas tworzenia całego modelu. Próbki mogą być losowane z rozkładu jednostajnego, bądź wykorzystując metody biorące pod uwagę ważność poszczególnych obserwacji [6].

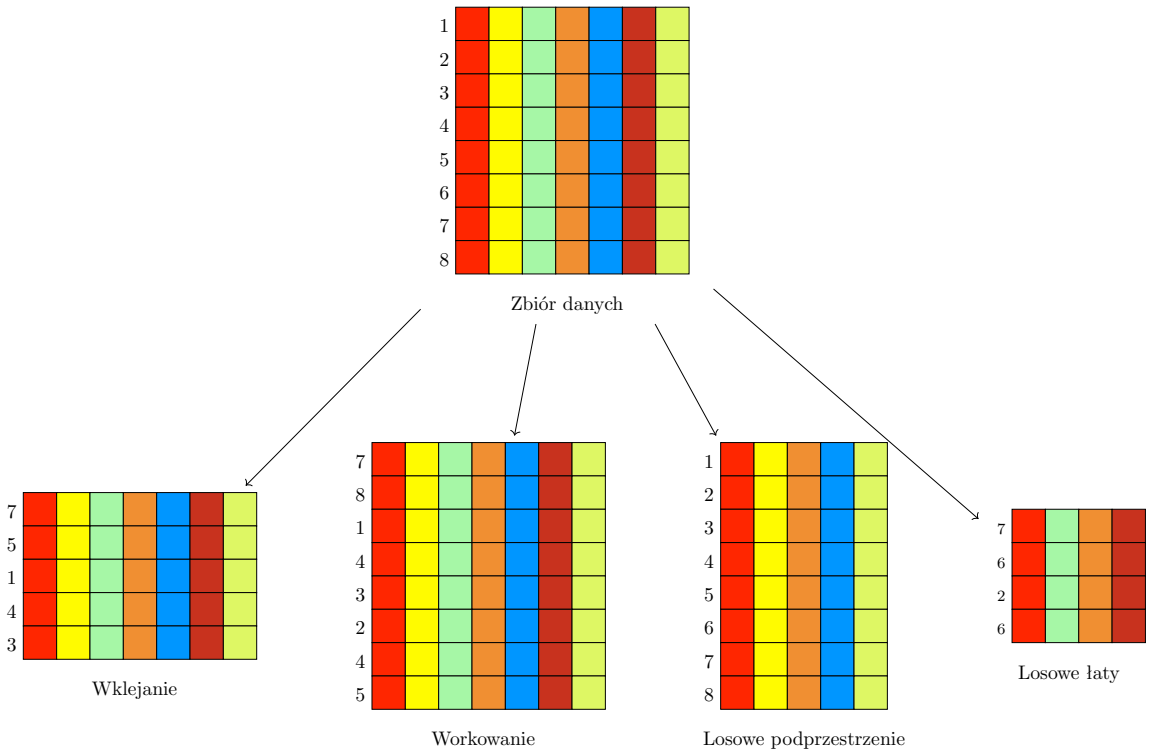
- Workowanie (*ang. Bagging*) – Polega na niezależnym losowaniu $K \leq N$ próbek ze zbioru $\mathcal{S}$ z powtórzeniami. Najczęściej jest wykorzystywany w przypadku małych zbiorów danych, kiedy chcemy wykorzystać maksymalnie dużą ilość obserwacji, lecz także wprowadzić losowość do kolejnych klasyfikatorów bazowych.

- Losowe podprzestrzenie (*ang. Random Subspaces*) – Metoda polegająca na wybraniu do każdego klasyfikatora bazowego wszystkich obserwacji, lecz losuje się tylko część atrybutów opisujące dane. Głównym celem tego zabiegu jest poprawa skuteczności poprzez zwiększenie różnorodności klasyfikatorów bazowych [19].

- Losowe łaty (*ang. Random Patches*) – Procedura łącząca metodę wklejania z losowymi podprzestrzeniami. Jej celem jest połączenie zalet obu metod i ostatecznie poprawienie skuteczności modelu z jednoczesną optymalizacją wykorzystywanego czasu oraz pamięci [15].

Na rysunku 1.3 na stronie 15 schematycznie przedstawiono wyżej omówione zasady działania poszczególnych procedur losowania. Należy zwrócić uwagę zarówno na powtarzające się lub nie numery wierszy, jak i na kolory kolumny symbolizujące odpowiednie zmienne z oryginalnego zbioru danych.

Warto zauważyć, że mimo tego, że powyższe metody były oryginalnie tworzone z myślą o wykorzystaniu drzewa decyzyjnego jako klasyfikatora bazowego, to nic nie stoi na przeszkodzie, aby wykorzystać inny model. Ten fakt został użyty na przykład w pracy A. Bahnsena [4], gdzie wykorzystując omawiane w sekcji 1.4.2 drzewa decyzyjne wrażliwe na koszt, stworzono lasy losowe wrażliwe na koszt.



Rysunek 1.3: Schemat działania procedur losowania. Źródło: Opracowanie własne.

Procedura tworzenia lasu losowego jest podzielona na dwie fazy [7]. Pierwsza z nich wykorzystuje workowanie do wygenerowania odpowiednich próbek dla klasyfikatorów bazowych. Następnie trenujemy zmodyfikowane drzewa decyzyjne, które na etapie tworzenia każdego kolejnego węzła w drzewie losują podzbiór zmiennych do wyboru podziału. Finalnie podczas predykcji każdy model dokonuje klasyfikacji i ostatecznym wynikiem modelu jest klasyfikacja większość głosów. Nie jest to jedyna możliwość agregowania wyników, przykładowo wykorzystywana w bibliotece *sklearn* w języku programowania Python, implementacja uśrednia wyniki prawdopodobieństwa z każdego klasyfikatora [9].

1.2.4 XGBoost

W ogólności proces trenowania modeli w uczeniu maszynowym polega na znalezieniu najlepszych parametrów modelu θ, które najlepiej pasują do danych treningowych ze zbioru $\mathcal{S}$ i ich prawdziwych wartości $\boldsymbol{y}$. W celu dokonania tego procesu definiujemy funkcję celu (*ang. objective*), którą w ogólnej postaci możemy zapisać jako

$$\text{Obj}(\theta) = J(\theta) + \Omega(\theta), \tag{1.6}$$

gdzie $J(\theta)$ to funkcja straty, a $\Omega(\theta)$ to wyraz odpowiadający za regularyzację, która nakłada odpowiednie kary na współczynniki modelu, aby zapobiec zbytniemu dopasowaniu do danych. Za funkcję straty najczęściej przyjmujemy w przypadku zadań regresyjnych błąd średniokwadratowy

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

gdzie $\hat{y}_i$ to predykcja modelu dla i-tej obserwacji i zależy ona od parametrów modelu θ, lub w przypadku zadań klasyfikacyjnych entropię krzyżową daną równaniem 1.5.

Algorytm XGBoost (*ang. Extreme Gradient Boosting*) to model, który w iteracyjny sposób tworzy kolejne drzewa decyzyjne typu CART (patrz sekcja 1.2.2). Na potrzeby algorytmu drzewa są zmodyfikowane i na liściu nie zawierają samej decyzji binarnej, lecz przypisują liczbę rzeczywistą, która daje znacznie bogatszą reprezentację [10]. Podobnie jak w przypadku lasu losowego wykorzystujemy wiele klasyfikatorów bazowych, a następnie dzięki wykorzystywanej reprezentacji możemy zapisać predykcję i-tej obserwacji jako

$$\hat{y}_i = \sum_{k=1}^{K} f_k(\boldsymbol{x}_i), \quad f_k \in \mathcal{F},$$

gdzie K oznacza liczbę drzew decyzyjnych, f_k funkcje z przestrzeni $\mathcal{F}$ wszystkich możliwych drzew CART. Warto zaznaczyć, że w przypadku zagadnienia regresyjnego predykcja y_i jest bezpośrednim wynikiem, natomiast dla problemów klasyfikacyjnych wykorzystuje się funkcję sigmoidalną (podobnie jak w regresji logistycznej) w celu zamiany wartości liczby rzeczywistej na predykcję z przedziału $[0, 1]$.

Korzystając z ogólnej postaci funkcji celu danej równaniem 1.6 funkcja $\mathrm{Obj}(\theta)$, którą będziemy optymalizować, przyjmuje następującą postać:

$$\mathrm{Obj}(\theta) = \sum_{i=1}^{N} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k),$$

gdzie $l(y_i, \hat{y}_i)$ to funkcja straty, przykładowo błąd średniokwadratowy lub entropia krzyżowa, a $\Omega(f_k)$ to regularyzacja nałożona na drzewo f_k.

Uczenie drzew decyzyjnych jest znacznie trudniejszym zadaniem niż standardowe zagadnienia optymalizacyjne, gdyż w tym przypadku nie jesteśmy w stanie wprost policzyć gradientu i wyliczyć kolejnych wartości współczynników. Zamiast tego użyjemy uczenia addytywnego, które w każdym kroku iteracji naprawia błędne klasyfikacje z poprzedniego kroku oraz tworzy kolejne drzewo. Przez $\hat{y}_i^{(t)}$ będziemy oznaczać predykcję dla i-tej obserwacji w kroku t. Zatem możemy zapisać kolejne predykcje w następujący sposób:

$$\hat{y}_i^{(0)} = 0,$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i),$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i),$$

$$\ldots$$

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i).$$

Podczas każdego z kroków iteracji będziemy chcieli minimalizować odpowiednią funkcję celu, która w tym przypadku ma postać

$$\mathrm{Obj}^{(t)}(\theta) = \sum_{i=1}^{N} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k).$$

Wykorzystując powyższe wzory, otrzymujemy następujące wyrażenie

$$\mathrm{Obj}^{(t)}(\theta) = \sum_{i=1}^{N} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \mathrm{const.}$$

Korzystając z rozwinięcia szeregu Taylora dla funkcji straty, otrzymujemy ogólny wzór:

$$\text{Obj}^{(t)}(\theta) = \sum_{i=1}^{N} \left[l(y_i, \hat{y_i}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + \text{const.} ,$$

gdzie:

$$g_i = \partial_{\hat{y_i}^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)}),$$
$$h_i = \partial^2_{\hat{y_i}^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)}).$$

Następnie po redukcji stałych, które są nieistotne z punktu widzenia optymalizacji, otrzymujemy

$$\sum_{i=1}^{N} [g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)] + \Omega(f_t).$$

Zauważmy, że nigdzie bezpośrednio nie korzystaliśmy z postaci funkcji straty, zatem w ten sposób jesteśmy w stanie zbudować model optymalizujący dowolną dwukrotnie różniczkowalną funkcję ciągłą. Taka uniwersalność powoduje, że algorytm ten znajduje zastosowanie w wielu zadniach z zakresu uczenia maszynowego, jak i innych problemów optymalizacyjnych.

1.3 Klasyfikacja wrażliwa na koszt

Metody klasyfikacji wrażliwej na koszt (*ang. Cost Dependent Classification*) są przykładem pierwszego rodzaju modeli wrażliwych na koszt. W przypadku tych metod trening zawierający informację o koszcie odbywa się dopiero po etapie stworzenia modelu zwracającego prawdopodobieństwa. Cały proces jest przedstawiony na rysunku 1.4 na stronie 18. Górna część diagramu przedstawia standardowy proces uczenia modelu predykcyjnego, natomiast dolna część reprezentuje schemat dokonywania predykcji. W celu wyznaczenia decyzji klasyfikatora najpierw estymujemy prawdopodobieństwa dla zbioru testowego, a następnie wykorzystując otrzymane wartości oraz koszt klasyfikacji, dokonujemy ostatecznej decyzji. Warto wspomnieć, że w celu dokonania treningu modelu klasyfikacji wrażliwej na koszt potrzebujemy dodatkowego podzbioru danych, który nie był wykorzystywany do uczenia standardowego modelu.

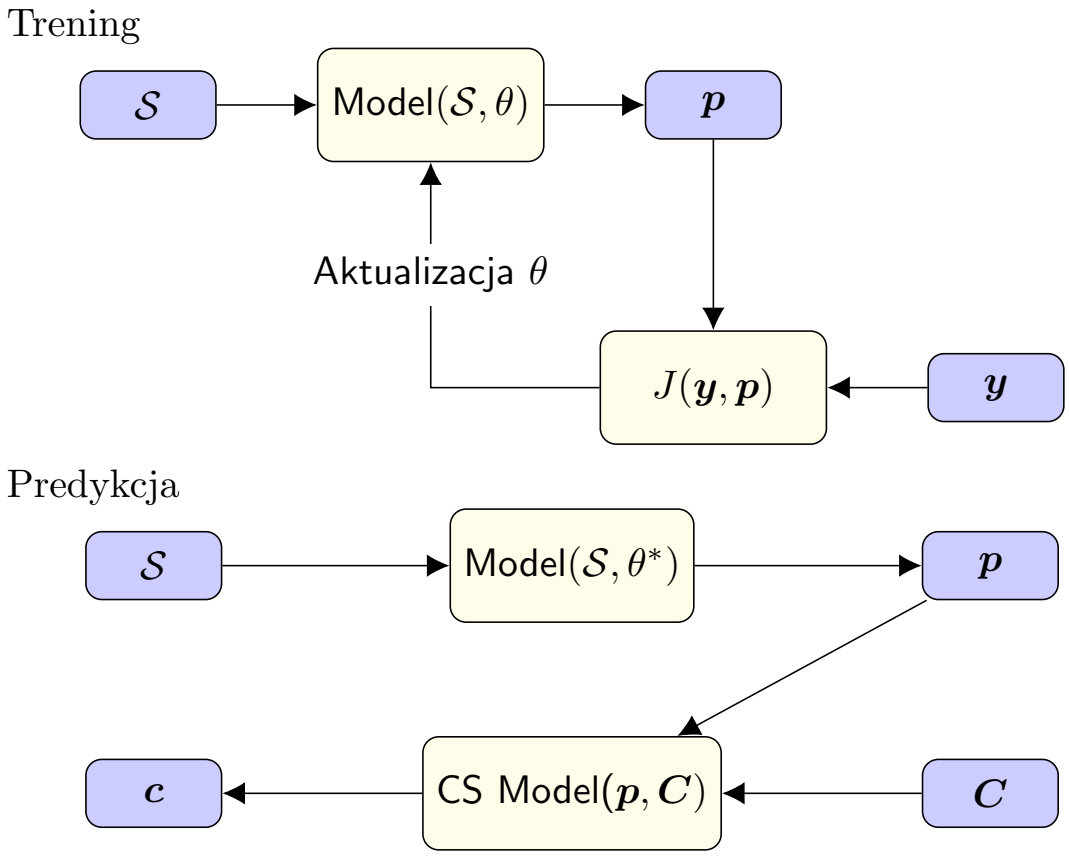
1.3.1 Optymalizacja progu

Metoda optymalizacji progu (TO, z angielskiego *threshold optimization*) jest popularną metodą wyznaczania odpowiedniego progu prawdopodobieństwa, powyżej którego wszystkie obserwacje oznaczamy jako pozytywnie zaklasyfikowane, a poniżej negatywnie. Może być ona wykorzystywana nie tylko do problemów wrażliwych na koszt, lecz do dowolnie zdefiniowanego zagadnienia, w którym potrzebne są zero-jedynkowe predykcje. Jej sformułowanie wygląda następująco

$$t^* = \arg\max_{t \in [0,1]} f(\boldsymbol{y}, \boldsymbol{c}),$$

gdzie:

- $f(\cdot, \cdot)$ – funkcja miary skuteczności modelu, która jako argumenty przyjmuje wektor prawdziwych wartości klasyfikacji oraz wektor binarnych predykcji, np. dokładność,



Rysunek 1.4: Schemat przedstawiający proces uczenia klasyfikatora wrażliwego na koszt. Źródło: Opracowanie własne na podstawie artykułu [16].

- $\boldsymbol{c} = (\mathbb{1}_{\{p_i > t\}})_{i=1}^n$ – wektor binarnych klasyfikacji modelu,
- t – wartość progu decyzyjnego.

W naszym przypadku funkcją f będzie funkcja oszczędności (1.3). Innymi słowy, będziemy szukać takiego progu decyzyjnego, który zmaksymalizuje wartość oszczędności.

1.3.2 Minimalizacja ryzyka bayesowskiego

Minimalizacja ryzyka bayesowskiego (BMR, z angielskiego *Bayesian Minimum Risk*) to model decyzyjny, którego celem jest zmierzenie oraz porównanie wartości prawdopodobieństw wystąpienia pewnych zdarzeń i kosztów związanych z podjęciem określonych decyzji [5]. Polega na przypisaniu odpowiedniej wartości reprezentującej ryzyko zaklasyfikowania danej obserwacji jako pozytywnej lub negatywnej, które definiujemy w następujący sposób

$$R(c_i = 1|\boldsymbol{x}_i) = L(c_i = 1|y_i = 1)P(y_i = 1|\boldsymbol{x}_i) + L(c_i = 1|y_i = 0)P(y_i = 0|\boldsymbol{x}_i),$$

$$R(c_i = 0|\boldsymbol{x}_i) = L(c_i = 0|y_i = 1)P(y_i = 1|\boldsymbol{x}_i) + L(c_i = 0|y_i = 0)P(y_i = 0|\boldsymbol{x}_i),$$

gdzie

- $P(y_i = 1|\boldsymbol{x}_i)$, $P(y_i = 0|\boldsymbol{x}_i)$ – oznaczają estymowane przez model prawdopodobieństwa zaklasyfikowania obserwacji jako odpowiednio pozytywna oraz negatywna klasa i $P(y_i = 0|\boldsymbol{x}_i) = 1 - P(y_i = 1|\boldsymbol{x}_i)$,

- $L(c_i = j|y_i = k)$ oraz $j, k \in \{0, 1\}$ – funkcja kosztu, która określa stratę, którą poniesiemy w zależności od wyniku poprawności klasyfikacji.

Zauważmy, że powyższe równania możemy zapisać jako

$$R(c_i = c|\boldsymbol{x}_i) = L(c_i = c|y_i = 1)P(y_i = 1|\boldsymbol{x}_i) + L(c_i = c|y_i = 0)P(y_i = 0|\boldsymbol{x}_i).$$

Rozważając przypadek binarny, możemy bez straty ogólności przyjąć, że

$$L(c_i = j|y_i = k) = C^{(i)}_{j,k},$$

zatem wzór na ryzyko przyjmuje postać

$$R(c_i = c|\boldsymbol{x}_i) = C^{(i)}_{c,1}P(y_i = 1|\boldsymbol{x}_i) + C^{(i)}_{c,0}P(y_i = 0|\boldsymbol{x}_i).$$

Ponadto warto zauważyć, że traktując y_i jako zmienną losową możemy zapisać ryzyko klasyfikacji danej obserwacji jako

$$R(c_i = c|\boldsymbol{x}_i) = E[C^{(i)}_{c,y_i}|\boldsymbol{x}_i].$$

Decyzja dotycząca danej obserwacji jest opisana następującą nierównością

$$c_i = 1 \Leftrightarrow R(c_i = 1|\boldsymbol{x}_i) \leq R(c_i = 0|\boldsymbol{x}_i)$$

i oznacza ona, że klasyfikujemy dany przykład jako pozytywny, jeżeli ryzyko takiej decyzji jest mniejsze niż zaklasyfikowania danej obserwacji jako negatywną. Po przeprowadzaniu odpowiednich przekształceń algebraicznych (wyprowadzone w dodatku B) oraz zakładając, że

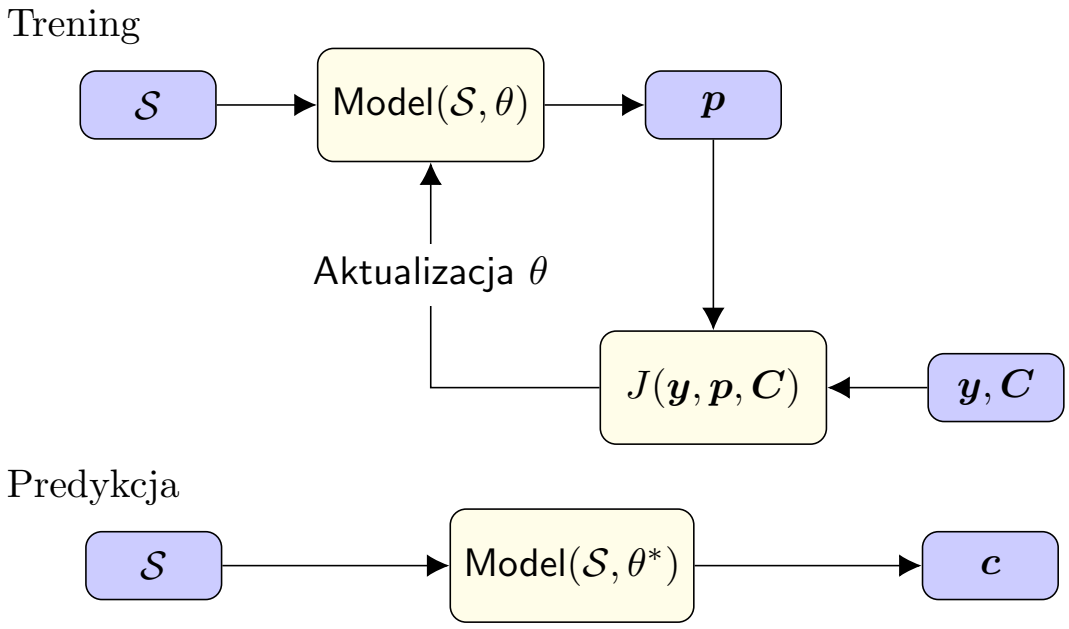
$$C^{(i)}_{0,1} - C^{(i)}_{1,1} - C^{(i)}_{0,0} + C^{(i)}_{1,1} > 0,$$

otrzymujemy następujący wzór na klasyfikację przykładu jako pozytywny

$$c_i = 1 \Leftrightarrow P(c_i = 1|\boldsymbol{x}_i) \geq \frac{C^{(i)}_{1,0} - C^{(i)}_{0,0}}{C^{(i)}_{0,1} - C^{(i)}_{1,1} - C^{(i)}_{0,0} + C^{(i)}_{1,0}}.$$

1.4 Trening wrażliwy na koszt

Drugą podgrupą metod wrażliwych na koszt jest trening wrażliwy na koszt (*ang. Cost Sensitive Trainig*). Metody te już na etapie treningu modelu biorą pod uwagę koszt związany z klasyfikacją danej obserwacji. Schemat uczenia modelu jest przedstawiony na wykresie 1.5 na stronie 20. Algorytm jako wejście otrzymuje zbiór danych treningowych, następnie dokonuje predykcji i na bazie prawdziwych odpowiedzi oraz kosztów wyznaczana jest skuteczność modelu. W następnej kolejności aktualizowane są wagi, a cały proces jest iteracyjnie powtarzany, aż do momentu osiągnięcia zadanego kryterium stopu. Predykcja w tym przypadku następuje w standardowy sposób.



Rysunek 1.5: Schemat przedstawiający proces uczenia modelu wrażliwego na koszt. Źródło: Opracowanie własne na podstawie artykułu [16].

1.4.1 Regresja logistyczna wrażliwa na koszt

Pierwszym modelem, który zostanie omówiony, jest regresja logistyczna wrażliwa na koszt. Kontynuując rozważania z podrozdziału 1.2.1, chcemy, aby funkcja straty standradowego modelu przyjmowała następujące wartości, które odpowiadają wartościom z macierzy kosztu

$$J_i^c(\theta) = \begin{cases} C_{1,1}^{(i)}, & \text{jeżeli } y_i = 1 \text{ i } h_\theta(\boldsymbol{x_i}) \approx 1, \\ C_{0,0}^{(i)}, & \text{jeżeli } y_i = 0 \text{ i } h_\theta(\boldsymbol{x_i}) \approx 0, \\ C_{1,0}^{(i)}, & \text{jeżeli } y_i = 0 \text{ i } h_\theta(\boldsymbol{x_i}) \approx 1, \\ C_{0,1}^{(i)}, & \text{jeżeli } y_i = 1 \text{ i } h_\theta(\boldsymbol{x_i}) \approx 0. \end{cases}$$

Przykładową funkcją, która zachowuje się w następujący sposób, jest

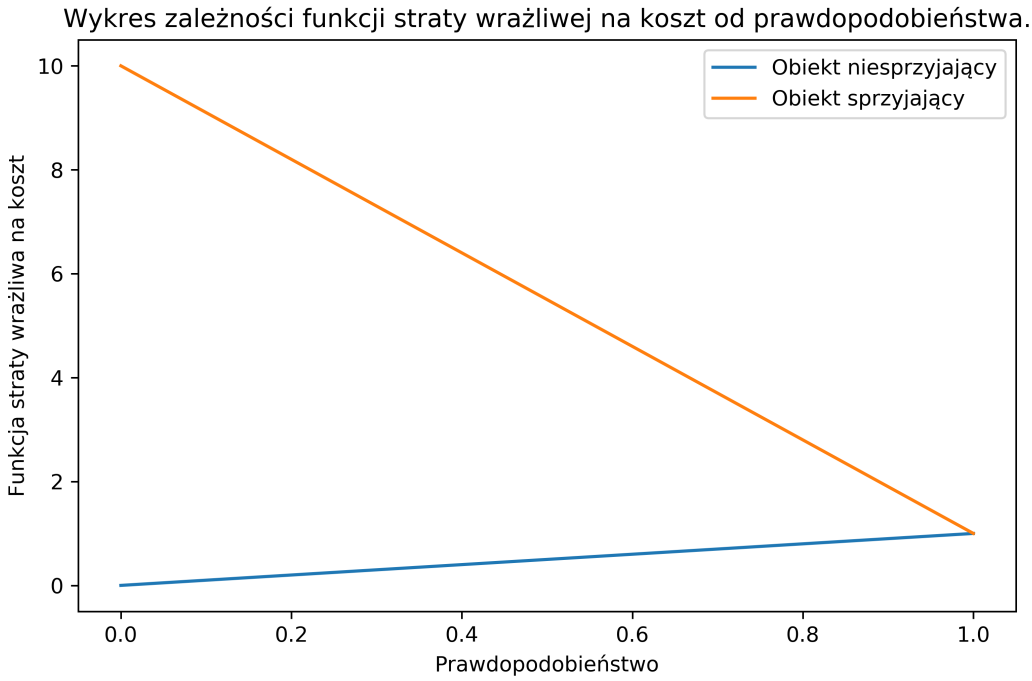
$$J_i^c(\theta) = y_i\Big(h_\theta(\boldsymbol{x_i})C_{1,1}^{(i)} + (1 - h_\theta(\boldsymbol{x_i}))C_{0,1}^{(i)}\Big) + (1 - y_i)\Big(h_\theta(\boldsymbol{x_i})C_{1,0}^{(i)} + (1 - h_\theta(\boldsymbol{x_i}))C_{0,0}^{(i)}\Big).$$

Wykres funkcji $J_i^c(\theta)$ jest przedstawiony na wykresie 1.6 na stronie 21. Kolor niebieski przedstawia wykres dla próbki o prawdziwej klasie pozytywnej, natomiast kolor pomarańczowy dla negatywnej. Możemy zauważyć pożądaną przez nas asymetrię popełniania błędów oraz osiąganie przez funkcję odpowiednich wartości w granicach prawdopodobieństwa równego 0 oraz 1.

Ostatecznie funkcją, którą będziemy minimalizować, jest

$$J^c(\theta) = \sum_{i=1}^{N} J_i^c(\theta).$$

Podobnie jak w przypadku standardowej regresji logistycznej, wykorzystując algorytm optymalizujący, szukamy optymalnych współczynników modelu.



Rysunek 1.6: Wykres zależności wrażliwej na koszt funkcji straty od prawdopodobieństwa predykcji danej obserwacji. Kolor niebieski przedstawia wykres dla próbki o prawdziwej klasie pozytywnej, natomiast kolor pomarańczowy dla negatywnej. Wykres dla przykładowych wartości: $C^{(i)}_{1,1} = 1$, $C^{(i)}_{0,1} = 10$, $C^{(i)}_{1,0} = 1$, $C^{(i)}_{0,0} = 0$. Źródło: Opracowanie własne.

1.4.2 Drzewo decyzyjne wrażliwe na koszt

Analogicznie jak w przypadku regresji logistycznej należy wprowadzić kosztu klasyfikacji danej próbki do procesu powstawania drzewa decyzyjnego. Naturalnym miejscem do uzależnienia modelu od kosztu klasyfikacji jest moment podziału danego węzła. Do tej pory wszystkie miary zanieczyszczenia skupiały się na możliwe najlepszym rozróżnieniu próbek w kontekście minimalizacji liczby błędów klasyfikacji. Zamiast tego, w przypadku metody wrażliwej na koszt, naszym celem jest minimalizacja kosztu. W tym celu definiujemy następującą miarę zanieczyszczenia

$$I_c(\mathcal{Q}) = \text{Koszt bazowy}(\boldsymbol{y}^{\mathcal{Q}}, \boldsymbol{C}^{\mathcal{Q}}),$$

gdzie górny indeks $\mathcal{Q}$ oznacza odpowiedni podzbiór obserwacji, które wybieramy do wektora $\boldsymbol{y}$ oraz $\boldsymbol{C}$. Intuicja stojąca za takim sformułowaniem jest następująca. Klasyfikujemy wszystkie przypadki w rozważanym podziale jako pozytywne, następnie jako negatywne i sprawdzamy, która z decyzji miała mniejszy sumaryczny koszt. W momencie predykcji ostateczny werdykt, który może być tylko binarny, zależy od tego, która klasyfikacja minimalizuje sumaryczny koszt na liściu.

Rozdział 2

Eksperyment

W celu porównania klasycznych oraz wrażliwych na koszt metod predykcyjnych przeprowadzono eksperyment na danych dotyczących detekcji oszustw na kartach płatniczych. Istotą doświadczenia było sprawdzenie, czy metody wrażliwe na koszt dają lepsze rezultaty niż standardowe modele. Jeżeli tak, to czy dzieje się tak dla każdej z metod, czy tylko dla niektórych. Ponadto, jeżeli nastąpi poprawa wyników względem oszczędności, to czy dzieje się to kosztem skuteczności w sensie standardowych miar.

Pierwsza część rozdziału dotyczy omówienia zawartości oraz pochodzenia zbioru danych. Dalej zostanie przedstawiona metodologia przeprowadzania eksperymentu. Rozważania zostaną zakończone omówieniem otrzymanych wyników oraz wyciągniętych wniosków.

2.1 Dane

Do eksperymentu został wykorzystany zbiór danych *Credit Card Fraud Detection*[1], który składa się z transakcji zawartych europejskimi kartami kredytowymi w ciągu dwóch dni we wrześniu 2013 roku. Dane były zebrane na potrzeby badań naukowych grupy Worldline and the Machine Learning Group[2] z Université Libre de Bruxelles. Zbiór zawiera 284,807 transakcji w tym zaledwie 492 oszustwa. Tabela składa się z 30 kolumn, w tym 28 z nich to zanonimizowane zmienne numeryczne, które były wcześniej poddane transformacji PCA (*ang. Principal Component Analysis*), a pozostałe dwie kolumny to informacje dotyczące godziny oraz kwoty transakcji. Ponadto wśród danych nie ma brakujących wartości. Podczas modelowania pominiemy zmienną czasową, ponieważ sama w sobie nie zawiera ona istotnej informacji, a tworzenie znaczących zmiennych nie jest istotą przeprowadzanego eksperymentu.

Mimo tego, że dane zostały poddane anonimizacji, to na podstawie artykułu A. Bahnsena możemy domyślać się, z jakimi zmiennymi mieliśmy do czynienia przed transformacjami [5]. Podczas procesu dokonywania transakcji standardowo zbierane są:

- data dokonania transakcji,
- numer konta,
- numer karty,
- typ transakcji (płatność internetowa, płatność stacjonarna, wypłata z bankomatu),

[1] Źródło: `https://www.kaggle.com/mlg-ulb/creditcardfraud`

[2] Strona grupy Worldline and the Machine Learning Group: `http://mlg.ulb.ac.be`

– kwota,

– ID beneficjenta transakcji,

– grupa beneficjenta transakcji (przykładowo linie lotnicze, hotel, wypożyczalnia samochodów itp.),

– kraj dokonania transakcji,

– kraj zamieszkania właściciela karty,

– typ karty (przykładowo VISA, MasterCard itp.),

– wiek klienta,

– płeć klienta,

– bank klienta,

– historyczna informacja czy dana transakcja była oszustwem.

Na podstawie wymienionych informacji tworzy się agregaty czasowe bazujące na historii, których celem jest zebranie informacji dotyczących nawyków zakupowych danego klienta [1]. Stworzenie takiej zmiennej rozpoczyna się od zgrupowania transakcji z pewnego odcinka czasowego najpierw według id klienta lub karty kredytowej, a następnie przez typ transakcji, beneficjenta transakcji, kategorię beneficjenta lub kraj. Proces kończy się agregacją powstałych grup poprzez zsumowanie wartości tych transakcji lub zliczenie ich ilości. Przykładowymi zmiennymi, które powstają w tym procesie, są ilość transakcji dla tego samego klienta w ciągu ostatnich 6 godzin, średnia wartość transakcji w sklepach spożywczych dla danej karty kredytowej z ostatniego tygodnia.

Na podstawie wiedzy dotyczącej modelowania detekcji oszustw z artykułu A. Bahnsena [4] w tabeli 2.1 na stronie 25 definiujemy wartości kosztu dla tego eksperymentu. Poniżej krótkie objaśnienie.

- Wartość $C_a = 1$ to stały koszt administracyjny podjęcia sprawy przez analityka, który sprawdza, czy dana obserwacja jest faktycznie oszustwem, niezależnie od tego, jaki był jego ostateczny werdykt.
- Amt_i to wartość transakcji, jaką utracimy w przypadku niewskazanie fałszywej transakcji jako oszustwa.
- Zerowa wartość kosztu dla normalnej obserwacji, która była prawidłowo wskazana, pochodzi z braku konieczności podjęcia inwestygacji oraz strat z tego wynikających.

2.2 Opis eksperymentu

Inspiracja do eksperymentu pochodzi z pracy A. Bahnsena dotyczącej porównania metod wrażliwych na koszt dla pięciu rzeczywistych zbiorów danych [4]. Dane pochodzą z czterech różnych dziedzin zastosowań: detekcja oszustw na kartach kredytowych, prognoza rezygnacji, ryzyko kredytowe oraz marketing bezpośredni. Celem pracy było porównanie standardowych metod oraz metod wrażliwych na koszt na przykładzie innego zbioru

	Stan pozytywny $y_i = 1$	Stan negatywny $y_i = 0$
Predykcja pozytywna $c_i = 1$	$C^{(i)}_{1,1} = \mathrm{C_a} = 1$	$C^{(i)}_{1,0} = \mathrm{C_a} = 1$
Predykcja negatywna $c_i = 0$	$C^{(i)}_{0,1} = \mathrm{Amt_i}$	$C^{(i)}_{0,0} = 0$

Tabela 2.1: Macierz kosztu eksperymentu.

danych. Do eksperymentu wykorzystano regresję logistyczną, drzewo decyzyjne, las losowy, XGBoost oraz drzewo decyzyjne wrażliwe na koszt. Ponadto standardowe modele zostały rozszerzone metodą optymalizacji progu (TO) oraz minimalizacji ryzyka bayesowskiego (BMR). Modyfikacją, w stosunku do oryginalnego eksperymentu, było użycie algorytmu XGBoost, który dotychczas w wielu standardowych zastosowaniach uzyskiwał znacznie lepsze wyniki od reszty modeli uczenia maszynowego.

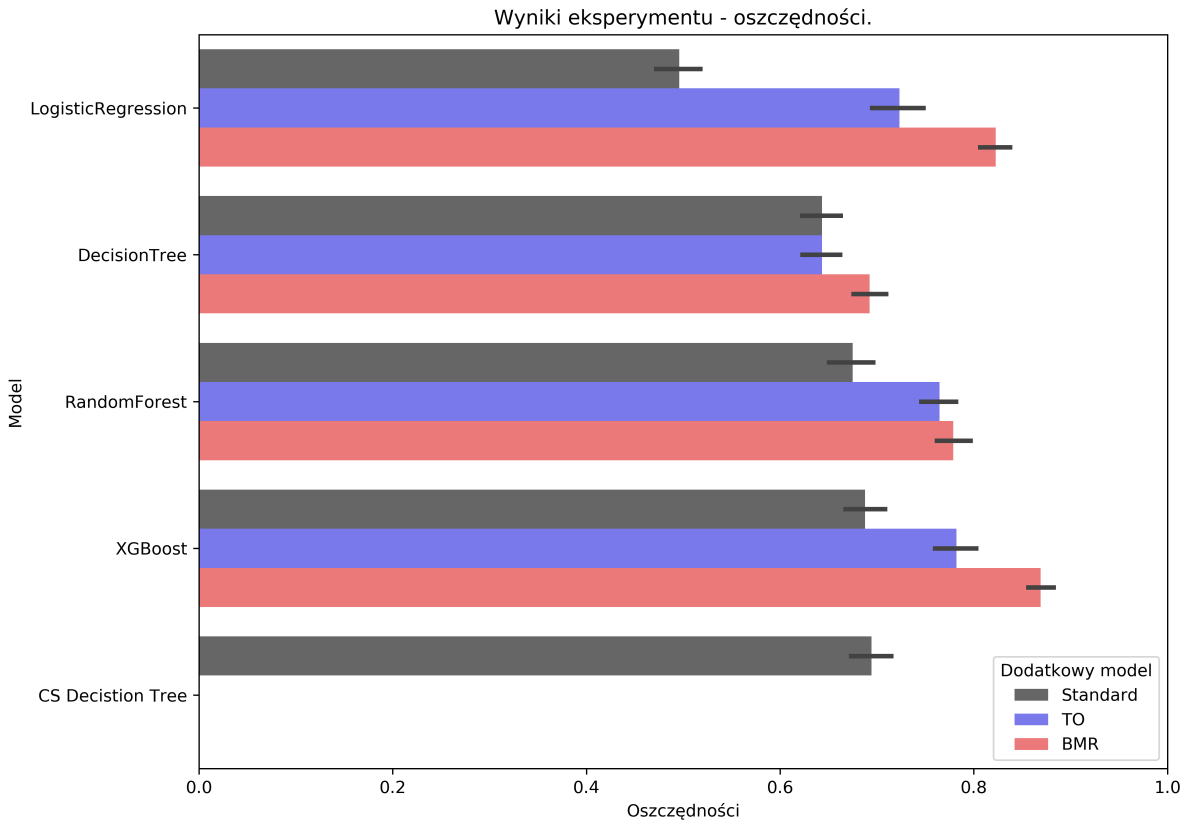
W celu przeprowadzenia eksperymentu zbiór danych został pięćdziesięciokrotnie podzielony w proporcjach 50:17:33 na odpowiednio zbiór treningowy, walidacyjny oraz testowy. Losowanie nowych podziałów zbioru miało na celu zmniejszenie ryzyka, że wynik będzie zależny od wylosowanej próbki. Ponadto wykorzystano tę technikę zamiast standardowej warstwowej walidacji krzyżowej (*ang. Stratified Cross Validation*) z uwagi na zbyt małą próbkę testową, która powstaje w przypadku wykorzystania walidacji krzyżowej. Następnie wytrenowano wszystkie modele standardowe oraz drzewo decyzyjne wrażliwe na koszt na zbiorze treningowym. Dla modelu XGBoost wykorzystano zbiór walidacyjny do procesu wczesnego zatrzymywania (*ang. Early Stopping*), natomiast dla modeli BMR oraz TO wykorzystano ten zbiór jako treningowy. Następnie dla wszystkich modeli dokonano predykcji na zbiorze testowym i zmierzono skuteczność typowań, wykorzystując miarę oszczędności oraz F_1 Score. Ostatecznie wyniki z wszystkich pięćdziesięciu prób zostały uśrednione oraz obliczono wartości odchyleń standardowych.

Do implementacji skryptów wykorzystano język programowania Python wraz z bibliotekami *costcla*, *sklearn*, *pandas*, *numpy*, *matplotlib* oraz język programowania BASH. Kody źródłowe programu można znaleźć w dodatku C.

2.3 Wyniki

Wyniki eksperymentu dla miary oszczędności są zaprezentowane na wykresie 2.1 na stronie 26. Szare, niebieskie oraz czerwone słupki oznaczają dla pierwszych czterech modeli: regresji logistycznej (LogisticRegression), drzewa decyzyjnego (DecisionTree), lasu losowego (RandomForest) oraz XGBoosta, odpowiednio model standardowy, model optymalizacji progu oraz model wykorzystujący minimalizację ryzyka bayesowskiego. Dla modelu drzewa decyzyjnego wrażliwego na koszt (CS Decision Tree) posiadamy jedynie szary słupek, ponieważ model ten sam w sobie jest już wrażliwy na koszt i nie potrzebuje dodatkowych modyfikacji.

Możemy zauważyć, że wśród standardowych modeli oraz drzewa decyzyjnego wrażliwego na koszt najmniejszy wynik oszczędności uzyskała regresja logistyczna a największy las losowy, XGBoost oraz drzewo decyzyjne wrażliwe na koszt. Dodatkowo zwraca uwagę fakt, że wrażliwa na koszt wersja drzewa decyzyjnego przyniosła lepsze rezultaty niż standardowy model. W przypadku, gdy weźmiemy pod uwagę rozszerzone wersje modeli

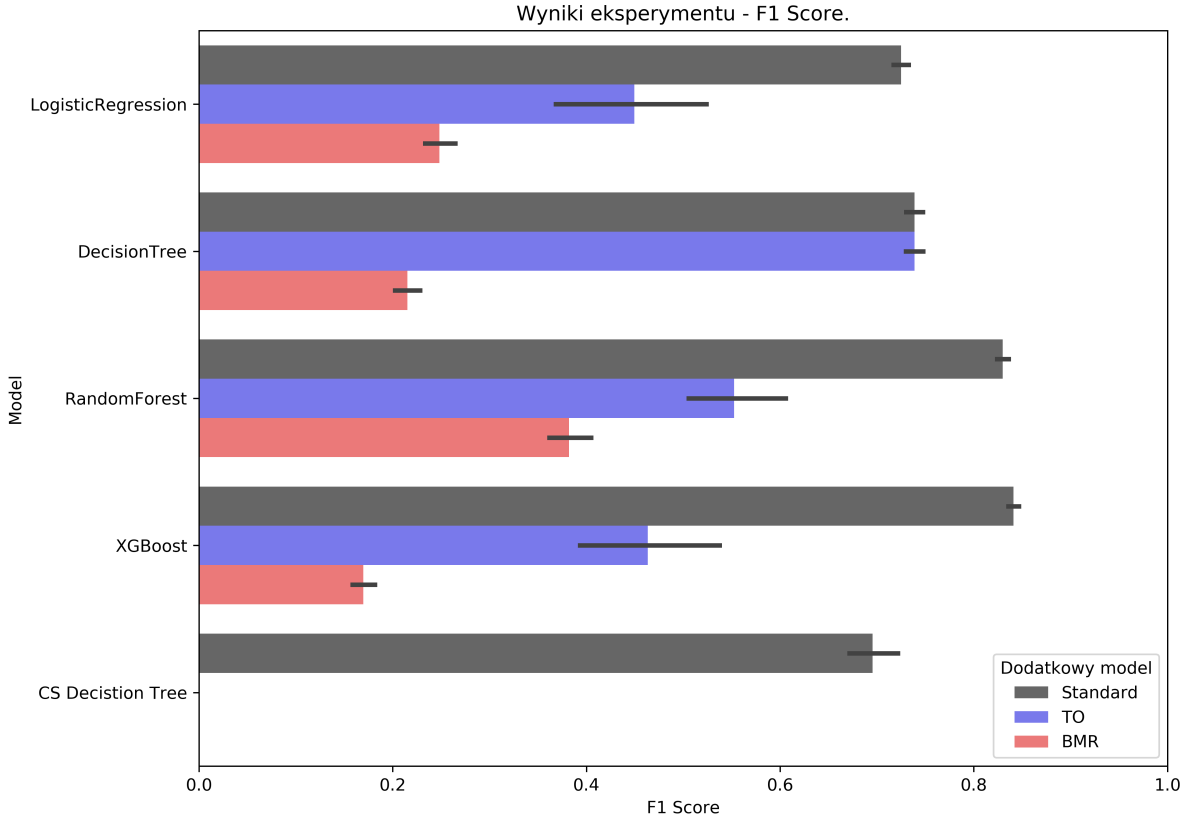


Rysunek 2.1: Wyniki eksperymentu dla miary skuteczności oszczędności. Szare, niebieskie oraz czerwone słupki oznaczają dla pierwszych czterech modeli: regresji logistycznej (LogisticRegression), drzewa decyzyjnego (DecisionTree), lasu losowego (RandomForest) oraz XGBoosta, odpowiednio model standardowy, model optymalizacji progu oraz model wykorzystujący minimalizację ryzyka bayesowskiego. Dla modelu drzewa decyzyjnego wrażliwego na koszt (CS Decision Tree) mamy jedynie jedną wartość. Wysokość słupka reprezentuje wartość średnią z 50 powtórzeń, a długość dołączonej kreski oznacza ± wartość odchylenia standardowego w odpowiednią stronę. Źródło: Opracowanie własne.

standardowych o klasyfikację wrażliwą na koszt, to zdecydowanie najlepszym modelem okazał się XGBoost w połączeniu z minimalizacją ryzyka bayesowskiego. Warto zauważyć, że w przypadku optymalizacji progu wartość oszczędności jest większa lub równa niż wynik uzyskany przez standardowy model. Dla minimalizacji ryzyka bayesowskiego zachodzi podobna sytuacja, gdzie w każdym przypadku wyniki są lepsze. Ponadto średnia wartość oszczędności modelu BMR w obrębie każdego z modeli standardowych jest najwyższa. Podsumowując, w przypadku tego eksperymentu, gdyby najważniejsza byłaby dla nas wartość oszczędności, to najlepszym wyborem byłby algorytm XGBoost z dodatkowym modelem minimalizacji ryzyka bayesowskiego. Co więcej, w każdym z przypadków, model wrażliwy na koszt poprawia wynik oszczędności.

W celu porównania zmiany skuteczności modeli względem standardowych miar przeanalizujemy wyniki eksperymentu dla F_1 Score, które są zaprezentowane na wykresie 2.2 na stronie 26. Oznaczenia są analogiczne jak do wykresu dotyczącego oszczędności. Możemy zauważyć, że dla standardowych wersji modeli w przypadku tej miary skuteczności najlepsze rezultaty osiągnął las losowy oraz XGBoost. Dodatkowo model drzewa decyzyjnego wrażliwego na koszt uzyskał w tym wypadku gorsze rezultaty niż standradowa

wersja. W przypadku klasyfikacji wrażliwej na koszt modele optymalizacji progu prawie zawsze pogarszały wyniki (z wyjątkiem drzewa decyzyjnego, gdzie wynik był taki sam jak bez modelu), a dla minimalizacji ryzyka bayesowskiego wynik zawsze był gorszy od standardowej wersji modelu. Intuicyjnie jest to zrozumiały rezultat, ponieważ nasz model przestał zwracać uwagę na transakcje, które miały małą wartość, nawet jeżeli przypadek ten był łatwy do wykrycia, a profilaktycznie zaczął sprawdzać duże transakcje, które mogły mieć nawet małe prawdopodobieństwa bycia oszustwem. W związku z tym modele zaczęły oszczędzać więcej pieniędzy kosztem ogólnej skuteczności modelu, przez to zwracają większą ilości fałszywie pozytywnych klasyfikacji.



Rysunek 2.2: Wyniki eksperymentu dla miary skuteczności F1-Score. Szare, niebieskie oraz czerwone słupki oznaczają dla pierwszych czterech modeli: regresji logistycznej (LogisticRegression), drzewa decyzyjnego (DecisionTree), lasu losowego (RandomForest) oraz XGBoosta, odpowiednio model standardowy, model optymalizacji progu oraz model wykorzystujący minimalizację ryzyka bayesowskiego. Dla modelu drzewa decyzyjnego wrażliwego na koszt (CS Decision Tree) mamy jedynie jedną wartość. Wysokość słupka reprezentuje wartość średnią z 50 powtórzeń, a długość dołączonej kreski oznacza $\pm$ wartość odchylenia standardowego w odpowiednią stronę. Źródło: Opracowanie własne.

Zestawiając uzyskane rezultaty z obu miar, możemy dojść do wniosku, że jeżeli najważniejszym i jedynym kryterium, które chcemy rozważać w ocenie jakości modelu, są oszczędności, to w przypadku tego zbioru danych najlepszym wyborem jest algorytm XGBoost z minimalizacją ryzyka bayesowskiego. Natomiast, jeżeli poza oszczędnościami, chcemy wziąć także pod uwagę ogólną jakość predykcyjną modelu, to wyniki nie jest aż tak jednoznaczny. W zależności od przypadku będziemy musieli wybierać pomiędzy ogólną skutecznością a oszczędnościami. Ciekawym wyborem w tym przypadku może być drzewo

decyzyjne wrażliwe na koszt, las losowy lub XGBoost w wersji standardowej. Dodatkową informacją, która mogłaby nam pomóc podjąć decyzję, może być ilość pozytywnych klasyfikacji z każdego modelu oraz ilość analityków zajmujących się sprawdzaniem oszustw, ponieważ może się okazać, że w wyniku zbyt małej ilości osób nie będziemy w stanie sprawdzić wszystkich wytypowanych spraw. W takiej sytuacji model, który typuje mniej transakcji do sprawdzania przy zachowaniu podobnej skuteczności dla obu miar, może okazać się najlepszym wyborem. Z drugiej strony, jeżeli taka informacja byłaby dostępna na początku modelowania, to wymusiłaby ona dodatkowe założenia oraz inne podejście do procesu doboru miary.

Ponadto pierwsze eksperymenty zostały wykonane również z wykorzystaniem regresji logistycznej wrażliwej na koszt, natomiast z nieznanych nam przyczyn (być może błędy implementacyjne) uzyskiwała ona ujemny wynik oszczędności oraz F_1 Score bliski zeru. Zatem uzyskane wyniki nie zostały opublikowane w podsumowaniu eksperymentu, ponieważ nie wnosiły one żadnej użytecznej informacji w kontekście przeprowadzanego wnioskowania. Ponadto poza tokiem przeprowadzonych prac został zrealizowany dodatkowy eksperyment na innym zbiorze danych i model ten uzyskiwał prawidłowe rezultaty, zatem wina może leżeć także po stronie naszego zbioru danych lub definicji macierzy kosztu.

Podsumowanie

Całość pracy rozpoczęło wprowadzenie problemu detekcji oszustw na kartach kredytowych oraz omówienie aktualnego podejścia do tworzenia systemów wykrywających oszustwa. Następnie omówiono rodzaje miar, które są standardowo wykorzystywane w statystyce oraz uczeniu maszynowym, a także miary, które stosuje się w problemach wrażliwych na koszt. Dalej opisano podstawy teoretyczne standardowych modeli predykcyjnych, ich modyfikacji wrażliwych na koszt oraz klasyfikatorów wrażliwych na koszt. W ostatniej części pracy przeprowadzono eksperyment, którego celem było porównanie wybranych klasyfikatorów na danych rzeczywistych dotyczących detekcji oszustw na kartach kredytowych.

Na podstawie przeprowadzonego doświadczenia ustalono, że na tym konkretnym zbiorze danych, dzięki wykorzystaniu metod wrażliwych na koszt, można zwiększyć oszczędności w problemie detekcji oszustw na kartach kredytowych. Najlepszym modelem pod tym względem okazał się XGBoost, który do tej pory nie był rozważanych w pracach A. Bahnsena, które dotyczyły podobnych zagadnień. Dodatkowo zauważono, że wykorzystanie klasyfikatorów wrażliwych na koszt znacząco obniża skuteczność modeli w sensie standardowych miar. W przypadku, potrzeby zachowania balansu pomiędzy tymi dwoma miarami, najlepszych wyborem byłby algorytm XGBoost, las losowy lub drzewo decyzyjne wrażliwe na koszt.

Na podstawie przeprowadzonych badań wydaje się, że bardzo ciekawym kierunkiem do dalszych rozważań mogą być metody typu *ensemble*, których klasyfikatorem bazowym byłoby drzewo decyzyjne wrażliwe na koszt. Takie działania zostały już podjęte dla lasów losowych w cytowanej wcześniej pracy A. Bahnsena [4]. Z uwagi na fakt, że algorytm XGBoost otrzymywał najlepsze rezultaty dla obu miar skuteczności oraz drzewo decyzyjne w wersji wrażliwej na koszt uzyskiwało lepsze rezultaty niż standardowa wersja modelu, daje to nadzieje, że XGBoost wrażliwy na koszt mógłby dawać jeszcze lepsze wyniki.

Dodatek A

Dokładność

W tej części udowodnimy następujący wzór

$$\text{Dokładność} = \frac{TP + TN}{TP + FP + FN + TN} = \alpha \cdot \text{Czułość} + (1 - \alpha) \cdot \text{Swoistość}. \tag{A.1}$$

Przypomnijmy, że

$$\alpha = \frac{TP + FN}{TP + FP + FN + TN},$$
$$1 - \alpha = \frac{TN + FP}{TP + FP + FN + TN},$$
$$\text{Czułość} = \frac{TP}{TP + FN},$$
$$\text{Swoistość} = \frac{TN}{TN + FP}.$$

Wstawiając do prawej strony równania A.1 odpowiednie wartości otrzymujemy

$$\frac{TP + FN}{TP + FP + FN + TN} \cdot \frac{TP}{TP + FN} + \frac{TN + FP}{TP + FP + FN + TN} \cdot \frac{TN}{TN + FP}.$$

Skracając ułamki otrzymujemy

$$\frac{TP}{TP + FP + FN + TN} + \frac{TN}{TP + FP + FN + TN} = \text{Dokładność}. \quad \square$$

Dodatek B

Minimalizacja ryzyka bayesowskiego

W tej części dodatku wyprowadzimy wzór

$$c_i = 1 \Leftrightarrow P(c_i = 1|\boldsymbol{x}_i) \geq \frac{C^{(i)}_{1,0} - C^{(i)}_{0,0}}{C^{(i)}_{0,1} - C^{(i)}_{1,1} - C^{(i)}_{0,0} + C^{(i)}_{1,0}}.$$

Przypomnijmy, że klasyfikujemy daną obserwację jako pozytywną, jeżeli prawdziwa jest nierówność

$$c_i = 1 \Leftrightarrow R(c_i = 1|\boldsymbol{x}_i) \leq R(c_i = 0|\boldsymbol{x}_i). \tag{B.1}$$

Ponadto w sekcji 1.3.2 definiowaliśmy ryzyko klasyfikacji w następujący sposób:

$$R(c_i = 1|\boldsymbol{x}_i) = C^{(i)}_{1,1}P(y_i = 1|\boldsymbol{x}_i) + C^{(i)}_{1,0}P(y_i = 0|\boldsymbol{x}_i),$$

$$R(c_i = 0|\boldsymbol{x}_i) = C^{(i)}_{0,1}P(y_i = 1|\boldsymbol{x}_i) + C^{(i)}_{0,0}P(y_i = 0|\boldsymbol{x}_i).$$

W celu skrócenia zapisu wprowadźmy następujące oznaczenie

$$P_k = P(y_i = k|\boldsymbol{x}_i), \quad k \in \{0, 1\}.$$

Korzystając z powyższych oznaczeń, definicji $R(c_i = 1|\boldsymbol{x}_i), R(c_i = 0|\boldsymbol{x}_i)$ oraz równania (B.1) otrzymujemy

$$C^{(i)}_{1,1}P_1 + C^{(i)}_{1,0}P_0 \leq C^{(i)}_{0,0}P_0 + C^{(i)}_{0,1}P_1.$$

Korzystając z własności

$$P_0 = 1 - P_1,$$

mamy

$$C^{(i)}_{1,1}P_1 + C^{(i)}_{1,0}(1 - P_1) \leq C^{(i)}_{0,0}(1 - P_1) + C^{(i)}_{0,1}P_1.$$

Następnie dokonując przekształceń algebraicznych, otrzymujemy

$$C^{(i)}_{1,0} - C^{(i)}_{0,0} \leq P_1 \cdot (C^{(i)}_{0,1} - C^{(i)}_{0,0} - C^{(i)}_{1,1} + C^{(i)}_{1,0}).$$

Zakładając, że $C^{(i)}_{0,1} - C^{(i)}_{0,0} - C^{(i)}_{1,1} + C^{(i)}_{1,0} > 0$ ostatecznie otrzymujemy

$$P_1 \geq \frac{C^{(i)}_{1,0} - C^{(i)}_{0,0}}{C^{(i)}_{0,1} - C^{(i)}_{0,0} - C^{(i)}_{1,1} + C^{(i)}_{1,0}}. \quad \square$$

Dodatek C

Skrypty

W tej części zostaną zaprezentowane skrypty, które były wykorzystane do przeprowadzenia eksperymentu.

Główna część eksperymentu była wykonywana w skrypcie *trainmodels.py*. Był on odpowiedzialny za wczytanie danych oraz konfiguracji, podział danych na odpowiednie podzbiory, a także rozdzielenie obliczeń na kilka wątków procesora. Wyniki z poszczególnych iteracji eksperymentu były zapisywane do plików *csv*, a następnie analizowane z wykorzystaniem notatnika *Jupyter Notebook*.

```
import os
import warnings

import numpy as np
import pandas as pd
import multiprocessing as mp

from datetime import datetime
from sklearn.model_selection import train_test_split

from src.utils import get_script_args, load_json, dict_union
from src.utils import create_cost_matrix, create_X_y
from src.utils import generate_models, generate_summary
from src.utils import train_standard_models, train_to_models,
    train_bmr_models

RANDOM_STATE = 42
OUTPUT_DIR = 'outputs'
CURRENT_OUTPUT_DIR = 'results' + datetime.now().isoformat()

warnings.filterwarnings('ignore')

np.random.seed(RANDOM_STATE)


def train_iteration(i, X, y, cost_matrix):
    print(f"Iteration: {i}.")
    models = generate_models()

    X_train, X_test, y_train, y_test, cost_matrix_train,
    cost_matrix_test = train_test_split(
        X, y, cost_matrix, train_size=0.5, stratify=y, random_state=i
    )
    X_val, X_test, y_val, y_test, cost_matrix_val, cost_matrix_test =
```

```
    train_test_split(
33          X_test, y_test, cost_matrix_test, train_size=0.33, stratify=
    y_test, random_state=i
34      )
35
36      standard_models = train_standard_models(X_train.values, y_train.
    values, cost_matrix_train, X_val.values, y_val.values, models)
37      threshold_optimized_models = train_to_models(X_val.values, y_val.
    values, cost_matrix_val, standard_models)
38      bmr_models = train_bmr_models(X_val.values, y_val.values,
    standard_models)
39
40      trained_models = dict_union(standard_models,
    threshold_optimized_models, bmr_models)
41      results = generate_summary(trained_models, X_test.values, y_test.
    values, cost_matrix_test)
42
43      filename = str(i) + '.csv'
44      filepath = os.path.join(OUTPUT_DIR, CURRENT_OUTPUT_DIR, filename)
45
46      results.to_csv(filepath, index=False)
47
48
49  def unpack_train_iterations(arguments):
50      i, X, y, cost_matrix = arguments
51      train_iteration(i, X, y, cost_matrix)
52
53
54  if __name__ == '__main__':
55      args = get_script_args()
56
57      config = load_json(args['config'])
58      n_iters = int(args['n_iters'])
59      df = pd.read_csv(args['data'])
60      X, y = create_X_y(
61          data=df,
62          class_column=config['ClassColumn'],
63          drop_columns=config['DropColumns']
64      )
65      cost_matrix = create_cost_matrix(
66          data=df,
67          fp_cost=config['FalsePositiveCost'],
68          fn_cost=config['FalseNegativeCost'],
69          tp_cost=config['TruePositiveCost'],
70          tn_cost=config['TrueNegativeCost']
71      )
72
73      os.mkdir(os.path.join(OUTPUT_DIR, CURRENT_OUTPUT_DIR))
74
75      pool = mp.Pool(mp.cpu_count())
76      results = pool.map(unpack_train_iterations, [(i, X, y, cost_matrix)
    for i in range(n_iters)])
```

Listing C.1: Skrypt *trainmodels.py* odpowiedzialny za główną część eksperymentu.

Zdecydowana większość kodu została zawarta w skrypcie *utils.py* i została oddzielona od poprzedniego programu, aby ułatwić zarządzanie częścią programistyczną projektu. Składa się on z funkcji służących między innymi do:

- wywoływania skryptu z poziomu linii komend,
- wczytywania plików *json* z konifugracją,
- generowania macierzy kosztu w odpowiednim formacie,
- trenowania modeli,
- tworzenia podsumowania wyników.

```
import json
import argparse
import xgboost

import numpy as np
import pandas as pd

from itertools import chain

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from costcla.metrics import cost_loss, savings_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from costcla.models import BayesMinimumRiskClassifier, ThresholdingOptimization
from costcla.models import CostSensitiveDecisionTreeClassifier, CostSensitiveLogisticRegression
from costcla.models import CostSensitiveRandomForestClassifier, CostSensitiveBaggingClassifier, \
    CostSensitivePastingClassifier, CostSensitiveRandomPatchesClassifier

N_JOBS = -1
RANDOM_STATE = 42

CI_MODELS = [LogisticRegression, DecisionTreeClassifier, RandomForestClassifier]
CST_MODELS = [CostSensitiveLogisticRegression, CostSensitiveDecisionTreeClassifier]
XGB_MODELS = [xgboost.XGBClassifier]
ECSDT_MODELS = [CostSensitiveRandomForestClassifier, CostSensitiveBaggingClassifier,
               CostSensitivePastingClassifier, CostSensitiveRandomPatchesClassifier]

def dict_union(*args):
    return dict(chain.from_iterable(d.items() for d in args))


def get_script_args():
    ap = argparse.ArgumentParser()
    ap.add_argument("-d", "--data", required=True, help="Path to data.")
    ap.add_argument("-c", "--config", required=True, help="Path to config.")
    ap.add_argument("-n", "--n_iters", required=False, default=10, help="Number of MC iterations.")
    args = vars(ap.parse_args())
```

```
    return args


def load_json(path):
    with open(path) as f:
        config = json.load(f)
    return config


def create_cost_matrix(data, fp_cost, fn_cost, tp_cost, tn_cost):
    # false positives, false negatives, true positives, true negatives
    def generate_cost(dataframe, cost):
        return dataframe[cost] if type(cost) == str else cost

    cost_matrix = np.zeros((data.shape[0], 4))

    cost_matrix[:, 0] = generate_cost(data, fp_cost)
    cost_matrix[:, 1] = generate_cost(data, fn_cost)
    cost_matrix[:, 2] = generate_cost(data, tp_cost)
    cost_matrix[:, 3] = generate_cost(data, tn_cost)

    return cost_matrix


def create_X_y(data, class_column, drop_columns):
    X = data.drop(drop_columns + [class_column], axis=1)
    y = data[class_column]
    return X, y


def _create_bmr_model(model, X_val, y_val, calibration=True):
    y_hat_val_proba = model.predict_proba(X_val)

    bmr = BayesMinimumRiskClassifier(calibration=calibration)
    bmr.fit(y_val, y_hat_val_proba)

    return model, bmr


def _create_threshold_optimized_model(model, X_val, y_val,
    cost_matrix_val, calibration=True):
    y_hat_val_proba = model.predict_proba(X_val)

    threshold_opt = ThresholdingOptimization(calibration=calibration)
    threshold_opt.fit(y_hat_val_proba, cost_matrix_val, y_val)

    return model, threshold_opt


def generate_models():
    models = {
        'CI-LogisticRegression': LogisticRegression(),
        'CI-DecisionTree': DecisionTreeClassifier(random_state=
    RANDOM_STATE),
        'CI-RandomForest': RandomForestClassifier(random_state=
    RANDOM_STATE),
        'CI-XGBoost': xgboost.XGBClassifier(random_state=RANDOM_STATE,
    verbosity=0),
```

```
        'CST-CostSensitiveDecisionTreeClassifier': CostSensitiveDecisionTreeClassifier()
    }
    return models


def train_standard_models(X_train, y_train, cost_matrix_train, X_val, y_val, models):
    trained_models = models.copy()
    # Standard model training
    for model in trained_models.values():
        model_type = type(model)
        print(model_type)
        if model_type in XGB_MODELS:
            model.fit(
                X_train, y_train,
                eval_set=[(X_val, y_val), (X_train, y_train)],
                eval_metric='aucpr',
                early_stopping_rounds=50,
                verbose=False
            )
        elif model_type in CST_MODELS or model_type in ECSDT_MODELS:
            model.fit(X_train, y_train, cost_matrix_train)
        elif model_type in CI_MODELS:
            model.fit(X_train, y_train)
        else:
            raise ValueError(f'Unknown model type: {model_type}.')

    return trained_models


def train_to_models(X_val, y_val, cost_matrix_val, models):
    trained_models = {}

    for name, model in models.items():
        model_type = type(model)
        if model_type in XGB_MODELS or model_type in CI_MODELS:
            for calibration in [True]:
                print(model_type, calibration)
                to_model = _create_threshold_optimized_model(model, X_val, y_val, cost_matrix_val, calibration=calibration)
                model_name = name + '-TO' #+ f'_{calibration}'
                trained_models[model_name] = to_model

    return trained_models


def train_bmr_models(X_val, y_val, models):
    trained_models = {}

    for name, model in models.items():
        model_type = type(model)
        print(model_type)
        if model_type in XGB_MODELS or model_type in CI_MODELS:
            for calibration in [True]:
                print(calibration)
                to_model = _create_bmr_model(model, X_val, y_val, calibration=calibration)
```

```
                model_name = name + '-BMR' #+ f'_{calibration}'
                trained_models[model_name] = to_model

    return trained_models


def _create_model_summary(model, name, X_test, y_test, cost_matrix_test)
    :
    standard_model_type = type(model)
    if standard_model_type == tuple:
        standard_model, extra_model = model
        extra_model_type = type(extra_model)
        if extra_model_type == BayesMinimumRiskClassifier:
            y_hat_proba = standard_model.predict_proba(X_test)
            y_hat = extra_model.predict(y_hat_proba, cost_matrix_test)
        elif extra_model_type == ThresholdingOptimization:
            y_hat_proba = standard_model.predict_proba(X_test)
            y_hat = extra_model.predict(y_hat_proba)
        else:
            raise ValueError(f'Unknown model type: {extra_model_type}.')
    elif standard_model_type in ECSDT_MODELS:
        y_hat = model.predict(X_test, cost_matrix_test)
    else:
        y_hat = model.predict(X_test)
    return {
        'Name': name,
        'Accuracy': accuracy_score(y_test, y_hat),
        'Precision': precision_score(y_test, y_hat),
        'Recall': recall_score(y_test, y_hat),
        'F1': f1_score(y_test, y_hat),
        'Cost': cost_loss(y_test, y_hat, cost_matrix_test),
        'Savings': savings_score(y_test, y_hat, cost_matrix_test)
    }


def generate_summary(trained_models, X_test, y_test, cost_matrix_test):
    results = pd.DataFrame(
        [_create_model_summary(
            model,
            name,
            X_test,
            y_test,
            cost_matrix_test
        ) for name, model in trained_models.items()
        ]
    )
    return results
```

Listing C.2: Skrypt *utils.py* zawierający funkcje pomocnicze do głównej części eksperymentu.

Eksperyment uruchamiano za pomocą skryptu *trainmodels.sh*, który wykonywał skrypt *trainmodels.py* z odpowiednimi argumentami.

```
#!/bin/bash

python3 train_models.py --data data/creditcard.csv --config config/
    creditcard-config-1.json --n_iters 50
```

Listing C.3: Skrypt *trainmodels.sh* odpowiedzialny za uruchamianie eksperymentu.

Bibliografia

[1] Bahnsen, A. C. Credit card fraud detection: Why theory doesn't adjust to practice. `https://www.youtube.com/watch?v=YCNkxaVDiA0`. SAS Analytics Conference, June 2, 2013, London, UK.

[2] Bahnsen, A. C., Aouada, D., Ottersten, B. Example-dependent cost-sensitive logistic regression for credit scoring. In *2014 13th International Conference on Machine Learning and Applications* (Dec 2014), pp. 263–269.

[3] Bahnsen, A. C., Aouada, D., Ottersten, B. Example-dependent cost-sensitive decision trees. *Expert Syst. Appl. 42*, 19 (Nov. 2015), 6609–6619.

[4] Bahnsen, A. C., Aouada, D., Ottersten, B. E. Ensemble of example-dependent cost-sensitive decision trees. *CoRR abs/1505.04637* (2015).

[5] Bahnsen, A. C., Stojanovic, A., Aouada, D., Ottersten, B. Cost sensitive credit card fraud detection using bayes minimum risk. In *2013 12th International Conference on Machine Learning and Applications* (Dec 2013), vol. 1, pp. 333–338.

[6] Breiman, L. Pasting small votes for classification in large databases and on-line. *Machine Learning 36*, 1 (Jul 1999), 85–103.

[7] Breiman, L. Random forests. *Machine Learning 45*, 1 (Oct 2001), 5–32.

[8] Brodersen, K. H., Ong, C. S., Stephan, K. E., Buhmann, J. M. The balanced accuracy and its posterior distribution. In *2010 20th International Conference on Pattern Recognition* (Aug 2010), pp. 3121–3124.

[9] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., Varoquaux, G. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (2013), pp. 108–122.

[10] Chen, T., Guestrin, C. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2016), KDD '16, ACM, pp. 785–794.

[11] Correa Bahnsen, A., Stojanovic, A., Aouada, D., Ottersten, B. Improving credit card fraud detection with calibrated probabilities. In *Proceedings of the 2014 SIAM International Conference on Data Mining. Pennsylvania, USA* (04 2014).

[12] Dal Pozzolo, A., Caelen, O., Le Borgne, Y.-A., Waterschoot, S., Bontempi, G. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications 41* (08 2014), 4915–4928.

[13] Dietterich, T. G. Ensemble methods in machine learning. In *Multiple Classifier Systems* (Berlin, Heidelberg, 2000), Springer Berlin Heidelberg, pp. 1–15.

[14] Elkan, C. The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2* (San Francisco, CA, USA, 2001), IJCAI'01, Morgan Kaufmann Publishers Inc., pp. 973–978.

[15] Louppe, G., Geurts, P. Ensembles on random patches. In *Machine Learning and Knowledge Discovery in Databases* (Berlin, Heidelberg, 2012), P. A. Flach, T. De Bie, and N. Cristianini, Eds., Springer Berlin Heidelberg, pp. 346–361.

[16] Moser, R. Fraud detection with cost-sensitive machine learning. `https://towardsdatascience.com/fraud-detection-with-cost-sensitive-machine-learning-24b8760d35d9`, 2019.

[17] Powers, D. Evaluation: From precision, recall and f-measure to ROC, informedness, markedness & correlation. *J. Mach. Learn. Technol 2* (01 2011), 2229–3981.

[18] Sheng, V. S., Ling, C. X. Thresholding for making classifiers cost-sensitive. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1* (2006), AAAI'06, AAAI Press, pp. 476–481.

[19] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 8 (Aug 1998), 832–844.

[20] Wikipedia contributors. Karta płatnicza – Wikipedia, wolna encyklopedia. `https://pl.wikipedia.org/wiki/Karta_p%C5%82atnicza`. [Online; dostęp 27-12-2019].

[21] Wikipedia contributors. Payment card – Wikipedia, wolna encyklopedia. `https://en.wikipedia.org/wiki/Payment_card`. [Online; dostęp 27-12-2019].