Przewidywanie oceny filmów
Link do projektu na githubie

Grzegorz Gawryał

1 Opis problemu

Celem projektu było przewidywanie średniej oceny filmów wystawionej przez użytkowników portalu internetowego IMDb na podstawie metadanych o filmie takich jak długość, budżet, reżyser, lista aktorów, opis itp. Dane zostały wybrane tak, aby można było estymować ocenę na podstawie samego pomysłu na film, więc nie mogły zawierać cech takich jak liczba głosów czy ocena wystawiona przez krytyków filmowych. Dane treningowe i testowe do projektu pochodzą z *kaggle'a*. Zbiór danych na stronie kaggle składał się z czterech plików:

- IMDb movies.csv, który zawierał główne metadane o filmie
- IMDb names.csv, zawierający dane o aktorach i innych osobach
- IMDb ratings.csv, w którym były bardziej szczegółowe dane o ocenach
- IMDb title_principals.csv, który opisywał ważność poszczególnych osób w każdym filmie

Wybór cech

Wszystkie cechy zostały wybrane z pliku IMDB movies.csv, ponieważ pozostałe pliki zawierały cechy nie mające dużego wpływu na przewidywanie oceny filmu lub uniemożliwiające przewidywanie oceny przed produkcją filmu. Uwzględnione zostały następujące cechy:

1. długość filmu (w minutach)
2. rok produkcji filmu
3. budżet w dolarach — w przypadku, gdy budżet w bazie danych był podany w innej wybranej walucie (euro, funt brytyjski, rupia indyjska, dolar kanadyjski, polski złoty), waluta została przeliczona zgodnie z kursem na początek 2021 roku na dolary
4. gatunek — w bazie danych było około 40 różnych gatunków filmowych, przy czym każdy film mógł mieć przypisanych kilka gatunków. Ze względu na to, że liczba różnych gatunków nie była różna, utworzone zostały osobne cechy zero jedynkowe dla każdego gatunku, i dany film miał na danej współrzędnej jedynkę wtw, gdy był odpowiedniego gatunku (one hot encoding).
5. reżyser (lub reżyserzy) — każdy film zawierał lisę reżyserów, najczęściej jednoelementową. Liczba różnych reżyserów była bardzo duża więc przetestowano wybranie pewnej liczby najpopularniejszych reżyserów i kodowanie ich jak gatunku, jednak po walidacji modelu zmieniono sposób kodowania na kodowanie docelowe (target encoding), które zostanie opisane niżej
6. aktorzy — każdy film w bazie danych zawierał listę najważniejszych aktorów (nieco ponad 10). Podobnie jak w przypadku aktorów, najpierw sprawdzono wyniki przy kodowaniu ustalonej liczby najczęściej grających aktorów, a później zmieniono kodowanie tak samo jak dla reżyserów
7. kraj — jak wyżej, target encoding
8. język — tak samo
9. opis filmu — baza danych zawierała krótki opisu filmu (około 30 słów), ucięty w przypadku gdy opis podany na stronie IMDb był za długi. Z opisu wycięto znaki interpunkcyjne, zastąpiono wszystkie litery małymi literami oraz zamieniono każde słowo na jego rdzeń (stemming) oraz zastosowano target encoding na tak powstałej liście słów z opisu

Filmy, o których informacje w bazie danych były niepełne (tzn. zawierały nulle) zostały pominięte, przez co liczba filmów w bazie zmniejszyła się z około 80 tyś. do 20 tyś. Na koniec, dane (łącznie z estymowaną oceną filmów) zostały ustandaryzowane, tzn. wyśrodkowane tak, aby średnia każdej kolumny wynosiła 0, a następnie przeskalowane tak, aby wariancja wynosiła 1.

Kodowanie docelowe (target encoding)

Przypuśćmy, że dana jest zmienna x przyjmująca wartości z pewnego zbioru klas $\{c_1, c_2, \ldots, c_C\}$ i zmienna estymowana y. Chcemy zakodować wartość x jako liczbę rzeczywistą tak, aby im większa ona była, tym większa była spodziewana wartość dla zmiennej y jej odpowiadającej. Mając dany zbiór obserwacji $x^{(1)}, \ldots, x^{(m)}$ i odpowiadający mu zbiór wartości $y^{(1)}, \ldots, y^{(m)}$ można policzyć dla każdej klasy występującej w zbiorze obserwacji średnią wartość zmiennych $y^{(i)}$, które odpowiadają zmiennym $x^{(i)}$ będącym ustalonego typu c_l i przewidywać dla nowych danych taką wartość y, jaką wyliczyliśmy dla odpowiedniej klasy. Aby zapobiec sytuacji, w której w zbiorze testowym będzie znajdować się nowa klasa, nie występująca w zbiorze obserwacji oraz aby uwzględnić wpływ całości zbioru obserwacji na pojedynczą obserwację, możemy dodać pewne wygładzenie.
Zatem, dla przychodzącej zmiennej $\widehat{x}$ przewidujemy:

$$\widehat{y} = \frac{\alpha\mu + \sum_i \mathbf{1}[x^{(i)} = \widehat{x}](y^{(i)})}{\alpha + |\{i : x^{(i)} = \widehat{x}\}|}$$

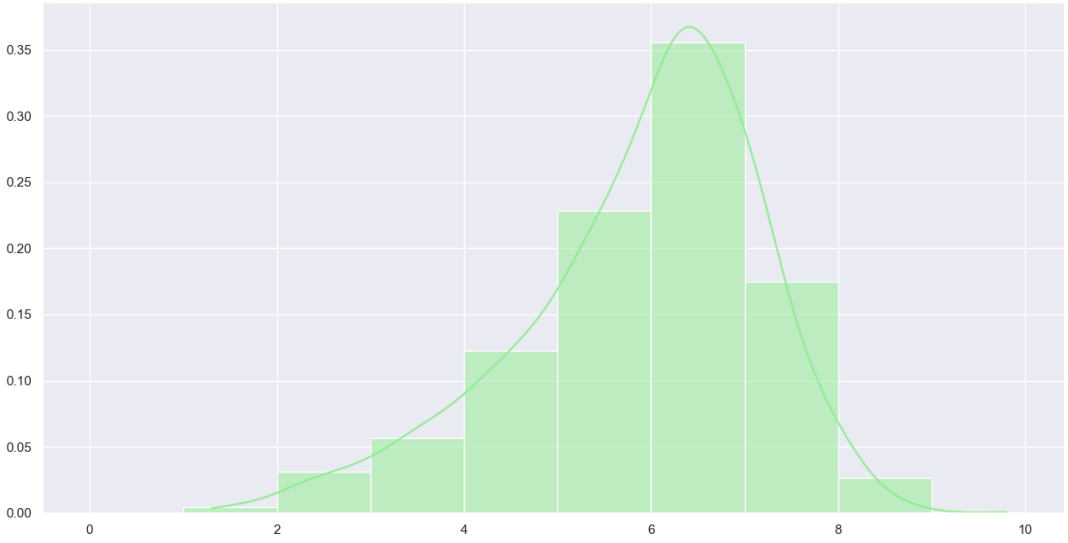
Gdzie $\mu = \frac{1}{m}\sum_i y^{(i)}$, a α jest parametrem wygładzenia — dla $\alpha = 0$ liczymy zwykłą średnią, a im większa α, tym większe znaczenie ma średnia wartość y na całym zbiorze obserwacji w stosunku do średniej wartości danej klasy.

Zapobieganie przeuczeniu

Kodując w ten sposób zmienne $x^{(i)}$ ze zbioru treningowego może pojawić się później problem przeuczenia modelu, który będzie się uczył na tych danych, zwłaszcza dla klas występujących bardzo rzadko. Przykładowo, dla klasy występującej tylko raz przypiszemy jej wartość $y^{(i)}$, którą ma zmienna $x^{(i)}$ mająca tę klasę. Może okazać się, że dodanie wygładzenia nie zmniejszy znacznie przeuczenia znacznie lub pogorszy wyniki, dlatego często wprowadza się tzw. *k-fold target encoding*.
Polega to na tym, że dla dzielimy zbiór treningowy na k mniej więcej równych grup i średnią dla każdej obserwacji w danej grupie wyznaczamy na podstawie obserwacji mających tę samą klasę, ale będących w pozostałych grupach. Dzięki temu zmienna $y^{(i)}$ nie ma wpływu na wartość, jaką dostanie zakodowana zmienna $x^{(i)}$. Szczególnym przypadkiem jest *leave one out target encoding*, w którym $k = m$, natomiast w praktyce często wykorzystuje się $k \in [4; 30]$

Rozkład średniej oceny filmów

Każdy film otrzymał średnio głosów, a rozkład średniej oceny filmów, przed standaryzacją, miał rozkład przypominający normalny, ze średnią około 5 i wariancją 10. Wykres poniżej przedstawia ten rozkład



Podział na dane

Zbiór treningowy, walidacyjny i testowy stanowiły odpowiednio $70\%, 15\%, 15\%$ wielkości całego zbioru danych, czyli miały po około $14000, 3000, 3000$ obserwacji. Ponieważ przetworzenie danych z pliku csv w sposób podany wyżej do ustandaryzowanej macierzy liczb zajmowało dość dużo czasu, wcześniej przygotowano 5 różnych podziałów na dane treningowe, walidacyjne i testowe w podanych proporcjach, a następnie wyniki każdego modelu uśredniono na 5 przebiegach algorytmu uczenia.

2 Rozważane modele

W projekcie przetestowane zostały następujące modele:

- Regresja liniowa w wersji z funkcjami jądrowymi
- Drzewa decyzyjne
- Lasy losowe
- Gradient boosting

2.1 Jądrowa regresja liniowa

W regresji liniowej w wersji jądrowej dla funkcji $\kappa(x, z) = \langle \phi(x), \phi(z) \rangle$ i macierzy K chcemy zminimalizować

$$J(\theta) = \frac{1}{2m} \sum \left(\theta^T \phi(x^{(i)}) - y^i \right)^2 + \frac{\lambda}{2} \sum \theta_j^2$$

Rozwiązanie analityczne możemy zapisać jako $\theta = \Big(K + \lambda I_m)^{-1} y$, natomiast nową przewidywaną wartość dla x jako $[\kappa(x, x^{(1)}), \ldots, \kappa(x, x^{(m)})]\theta$ W projekcie zostało zaimplementowane to rozwiązanie analityczne. Ponieważ utrzymuje ono macierz kwadratową rozmiaru 14000×14000, wymaga ono doś dużej ilości pamięci do policzenia, jednak czasowo działa dość szybko (czas rzędu kilkuset sekund na całym zbiorze treningowym).
Dla zbioru danych w projekcie m było zdecydowanie większe w porównaniu do k, więc bardziej naturalnym wyborem byłaby zwykła regresja liniowa, jednak nie jest możliwe użycie w niej wszystkich funkcji bazowych stanowiących jądro gaussowskie. W projekcie sprawdzono jądro gaussowskie, wielomianowe oraz liniowe. Jądra liniowe i gaussowskie, z dobranym parametrem $\gamma = 0.009$ nie wykazywały cech przeuczenia, natomiast jądro wielomianowe było bardzo podatne na przeuczenie i wymagało dobrania odpowiednio dużej stałej λ dla różnych jego parametrów.

2.2 Drzewo decyzyjne

Pojedyncze drzewo decyzyjne zostało zaimplementowane tak, aby mogło zostać wykorzystane później w lasach losowych czy gradient boostingu. Jego hiperparametrami są maksymalna głębokość, minimalna liczba wierzchołków w liściu oraz funkcja, jaką zwraca drzewo decyzyjne będąc w liściu (mediana lub średnia). Optymalną maksymalną głębokością drzewa było 8 i model uzyskał średni błąd kwadratowy wynoszący około 0.6 na danych walidacyjnych, natomiast na treningowych 0.47, co wskazywało na lekkie przeuczenie, jednak dodanie obcinania drzewa zwiększyłoby czas potrzebny do uczenia, a nie poprawiłoby wyników w pozostałych modelach korzystających z drzewa decyzyjnego.

Drzewo decyzyjne słabo radziło sobie z danymi kategorycznymi zakodowanymi zero jedynkowo takimi jak reżyser lub aktorzy, ignorując je. Po zmianie sposobu kodowania tych cech na kodowanie docelowe, drzewo poprawiło swoją dokładność przewidywania i tworzyło wierzchołki, w których punktami podziału były tak zakodowane cechy. Wyjątkiem była cecha gatunek, dla której drzewo już od początku radziło sobie dostatecznie dobrze i zmiana jego sposobu kodowania nie przyniosła poprawy.

2.3 Lasy losowe

Model lasów losowych składał się z 30 drzew, z których każde miało ograniczoną głębokość przez 8, dla każdego generowano próbkę boostrapową danych testowych wybierając tylko pewną liczbę cech. Model zgodnie z oczekiwaniem osiągał nieco lepsze wyniki niż pojedyncze drzewo oraz rozkład przewidywanych przez nie wartości y był bardziej stabilny, tzn. miał mniej ekstremów lokalnych.
Analogicznie jak w przypadku pojedynczego drzewa, model słabo sobie radził z danymi zakodowanymi zero jedynkowo,

a dodatkowym jego problemem był fakt, że do dla małego losowego zbioru kolumn z dość dużym prawdopodobieństwem trafiały prawie same cechy zakodowane zero jedynkowo, w których prawie wszystkie obserwacje miały zera, przez co zbiór wybieranych cech musiał być duży.
Ponieważ ostatecznie gatunek filmu pozostał zakodowany zero jedynkowo, więc liczbę wybieranych losowo cech ustalono na połowę wszystkich cech.

2.4 Gradient boosting

2.4.1 Opis teoretyczny

Zgodnie z ideą boostnigu, będziemy chcieli połączyć T słabych modeli dla regresji h_t w jeden silny h w następujący sposób:

$$h(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Dla pewnych wag α_t. W przeciwieństwie do algorytmu AdaBoost, nie będziemy korzystać ani zmieniać wag obserwacji.
Idea jest następująca:
W każdej iteracji będziemy poprawiać aktualny model $F_i(x) = \sum_{j=1}^{i} \alpha_j h_{p_j}(x)$, chcąc dodać pewien model ze zbioru słabych modeli z odpowiednią wagą, aby uzyskać $F_{i+1}(x) = \sum_{j=1}^{i+1} \alpha_j h_{p_j}(x)$
Chcemy tak dobrać α_{i+1}, p_{i+1}, aby zminimalizować oczekiwaną stratę, czyli

$$\sum_{k=1}^{m} L(y^{(k)}, F_{i+1}(x^{(k)})) = \sum_{k=1}^{m} L\Big(y^{(k)}, F_i(x^{(k)}) + \alpha_{j+1} h_{p_{j+1}}(x^{(k)})\Big)$$

Możemy patrzeć na składnik $\alpha_{j+1} h_{p_{j+1}}(x^{(k)})$ jak na krok gradientu w kierunku najwyższego spadku w celu znalezienia minimum — analogicznie jak w algorytmie spadku wzdłuż gradientu, gdzie α_{j+1} jest wielkością kroku
Możemy zatem policzyć ujemny gradient

$$-g_i(x^{(k)}) = -\frac{\partial L(y^{(k)}, F_i(x^{(k)}))}{\partial F_i(x^{(i)})}$$

w punkcie $F_i(x)$, a następnie wybrać słaby model regresji, który najlepiej przybliża tę wartość gradientu. Wybieramy zatem model minimalizujący $\sum_{k=1}^{m} \Big(-g_i(x^{(k)}) - \beta h_{p_{j+1}}(x^{(k)}) \Big)^2$, gdzie minimalizacja jest po zmiennych p_{j+1}, β
Następnie, znajdujemy wartość $\alpha_{m+1} = \text{argmin}_\alpha \sum_{k=1}^{m} L(y^{(k)}, F_m(x^{(k)}) + \rho h_{p_{j+1}}(x^{(k)}))$

Dla kwadratowej funkcji straty $L(y, z) = \frac{1}{2} \sum_{k=1}^{m} (y_i - z_i)^2$, jej gradient wynosi $y - z$, zatem algorytm się jeszcze bardziej upraszcza:

: LS_boost

$F_0(x) = \frac{1}{m} \sum_{k=1}^{m} y_k$
for $i = 1$ *to* MAX_ITERS **do**
  for $k = 1$ *to* m **do**
    $g_k = y_k - F_{i-1}(x_k)$
  $(\alpha_i, p_i) = \text{argmin}_{\alpha,j} \sum_{k=1}^{m} (g_k - \alpha h_{p_j}(x_k))^2$
  $F_i = F_{i-1} + \alpha_i h_{p_i}$

W praktyce, jeśli zbiorem słabych modeli regresji jest zbiór drzew decyzyjnych, to możemy dla uproszczenia pominąć minimalizację α_i i nadać jej ustaloną wartość (jak w spadku wzdłuż gradientu) i nauczyć drzewo tak, aby zminimalizowało ono kwadratową funkcję straty dla danych wejściowych $x^{(1)}, \ldots x^{(m)}$ i danych wyjściowych $-g_i(x^{(1)}), \ldots, -g_i(x^{(m)})$.
Oprócz kwadratowej funkcji straty, można rozważać inne, takie jak strata bezwzględna lub funkcję hubera. W pracy Jerome Friedmana [1] znajduje się opis implementacji algorytmów *LAD_TreeBoost* oraz *M_TreeBoost* korzystających odpowiednio z podanych funkcji strat oraz przy założeniu, że zbiór słabych modeli to drzewa decyzyjne.

2.4.2 Implementacja

W projekcie sprawdzono działanie wszystkich trzech wariantów funkcji straty. W modelu wykonywano 100 iteracji z parametrem $\alpha = 0.05$ i maksymalnej wysokości drzewa wynoszącej 4 lub 5. Ponadto, w każdej iteracji korzystano wyłącznie z losowego zbioru 60% wszystkich obserwacji.

3 Uzyskane wyniki

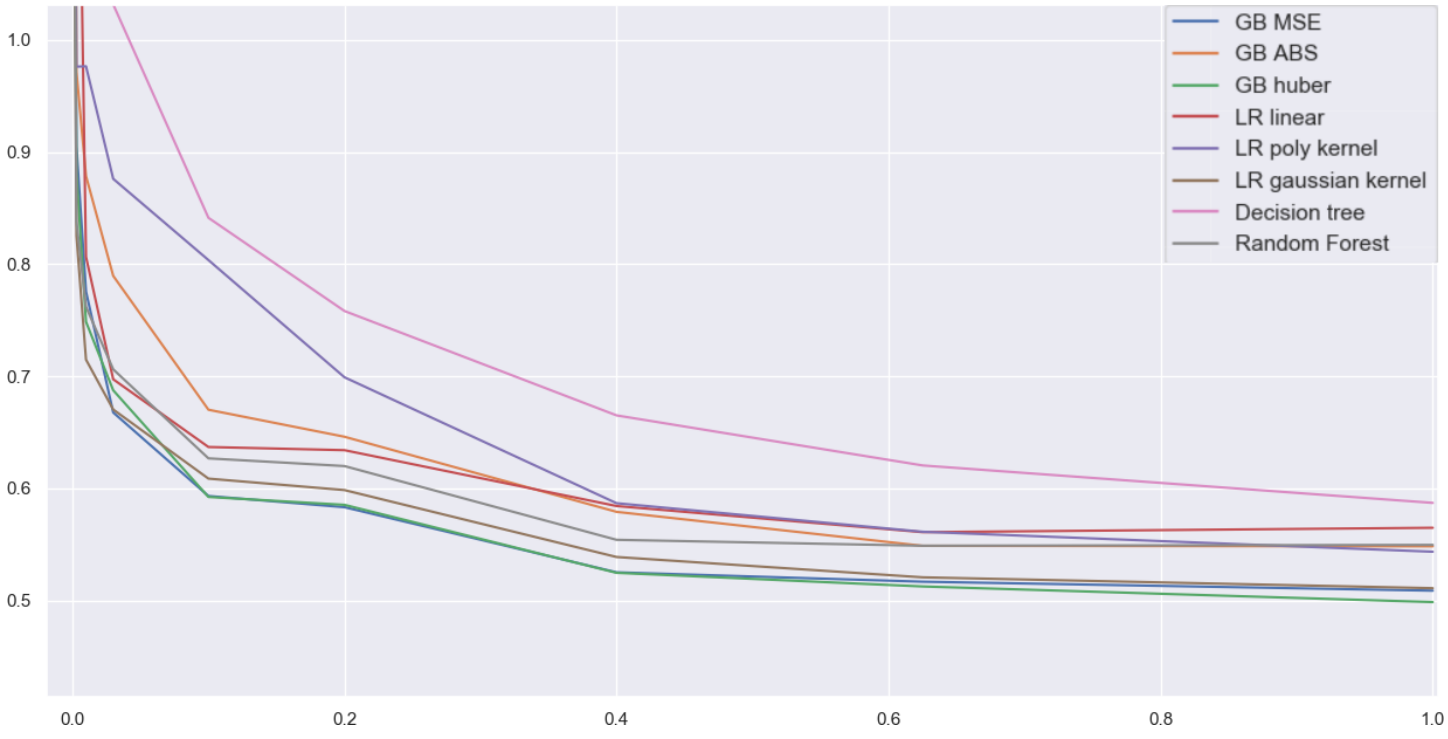
Wszystkie wyniki oraz współrzędne y na krzywych uczenia zostały uśrednione na 5 przebiegach algorytmów z różnym podziałem na zbiory treningowy, walidacyjny i testowy.
Tabela przedstawia średni błąd kwadratowy uzyskany przez wszystkie modele na zbiorze treningowym i testowym po nauce na całym zbiorze treningowym oraz miarę R2 na zbiorze testowym.

Model	MSE zbiór treningowy	MSE zbiór testowy	miara R2 na zbiorze testowym
Regresja liniowa	0.56	0.56	0.44
Regresja liniowa z jądrem wielomianowym	0.49	0.54	0.45
Regresja liniowa z jądrem gaussowskim	0.49	0.51	0.49
Drzewo decyzyjne	0.48	0.59	0.42
Las losowy	0.47	0.55	0.46
Gradient boosting z kwadratową stratą	0.47	0.51	0.49
Gradient boosting z bezwzględną stratą	0.43	0.55	0.46
Gradient boosting z funkcją Hubera	0.46	0.50	0.50

Przed zmianą sposobu kodowania zmiennych kategorycznych na kodowanie docelowe, najlepszy z modeli uzyskiwał średni błąd kwadratowy wynoszący około 0.6

3.1 Krzywe uczenia

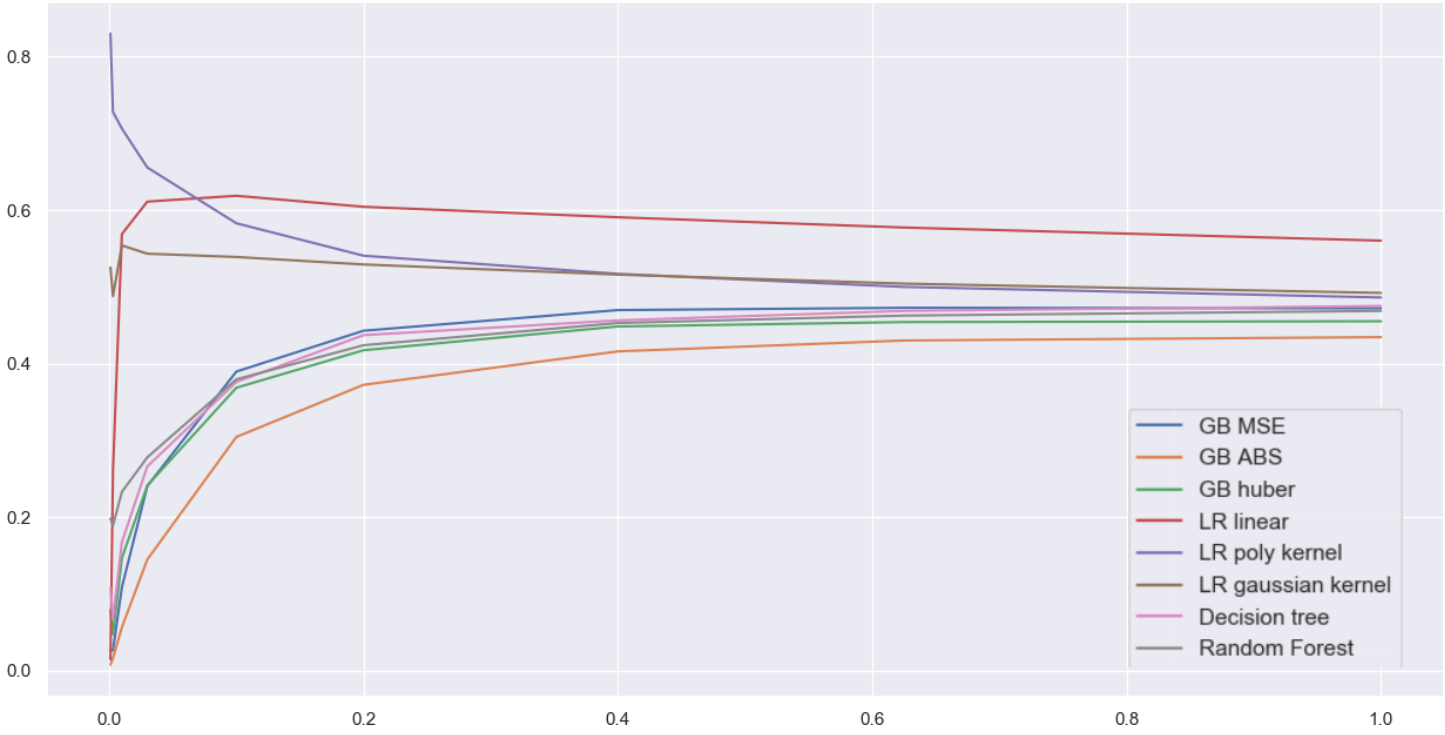
Pierwszy wykres prezentuje średni błąd kwadratowy na danych walidacyjnych uzyskany przez poszczególne modele:



Rysunek 1: Średni błąd kwadratowy ma danych walidacyjnych

Jak widać, wszystkie modele uczą się w bardzo podobnym tempie.

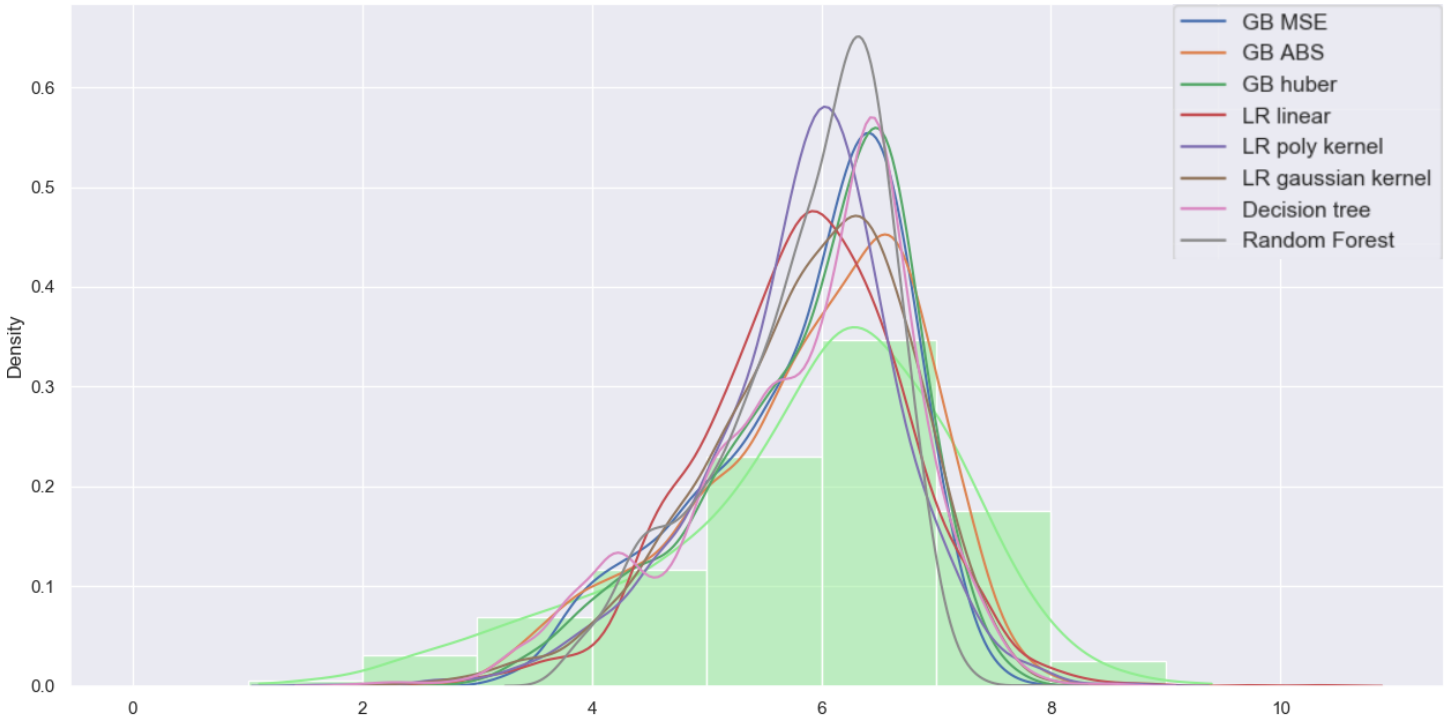
Kolejny wykres przedstawia krzywą średniego błędu kwadratowego na danych treningowych w zależności od frakcji wielkości tego zbioru



Rysunek 2: Średni błąd kwadratowy ma danych treningowych

Tu szczególną uwagę przykuwają wykresy czerwony, brązowy i fioletowy (regresje liniowe), które bardzo szybko stabilizują błąd na danych treningowych, który ostatecznie jest podobny do błędu na danych walidacyjnych. Ponadto, krzywa fioletowa jako jedyna jest malejąca. Wynika to z tego, że stała regularyzacji w regresji z jądrem wielomianowym ma bardzo duży wpływ na uzyskane wyniki oraz została dobrana jako stała na podstawie całego zbioru treningowego (a nie jako funkcja od jego wielkości).

Jasnozielone słupki i krzywa KDE (kernel density estimate) przedstawiają rozkład danej estymowanej na zbiorze walidacyjnym. Pozostałe krzywe przedstawiają rozkład danych estymowanych, które przewidywał każdy z modeli.



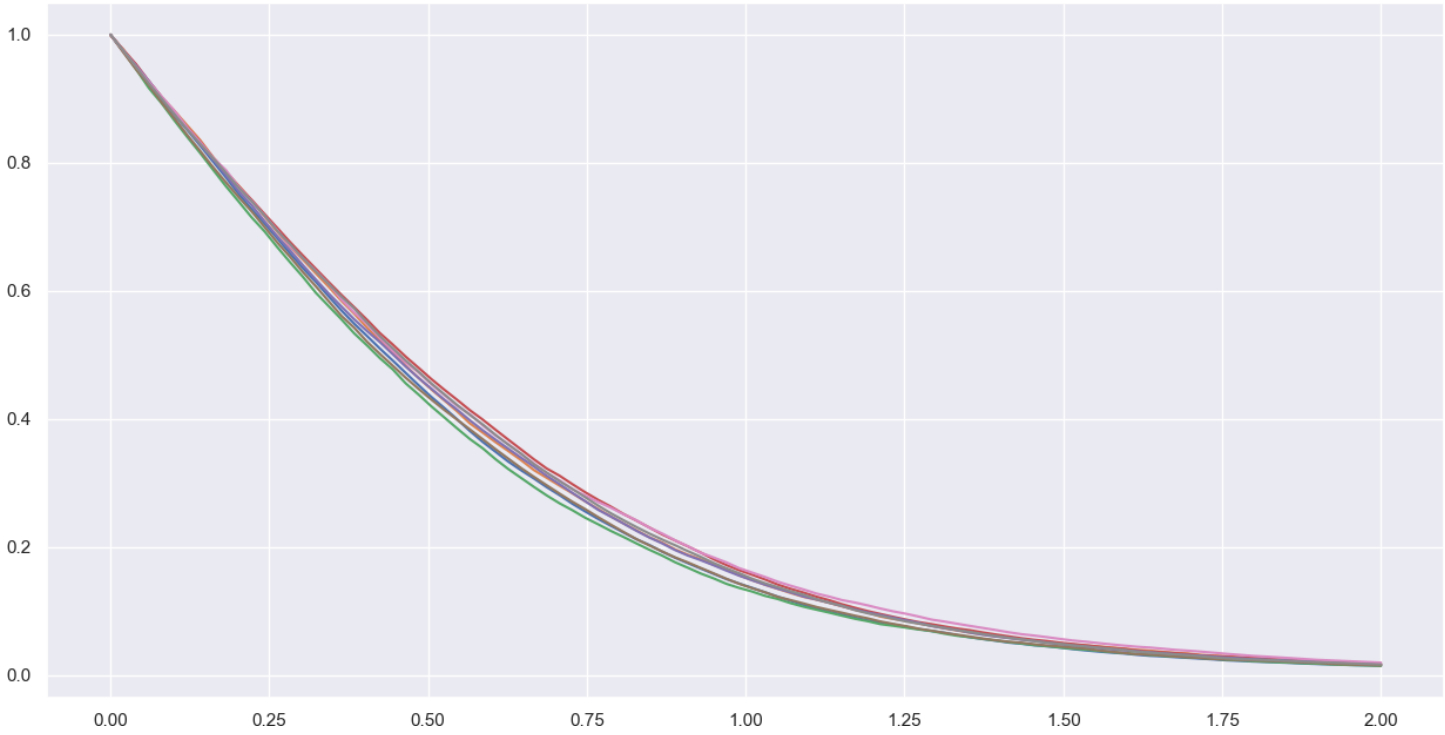
Rysunek 3: Rozkład przewidywanych danych na zbiorze testowym

Można zauważyć, że zmienna estymowana przez każdy z modeli ma mniejszą wariancję od prawdziwej danej. Modele są mniej skłonne wystawiania wyższych lub niższych ocen niż średnia ocena, ale kształt krzywej dość dobrze oddaje rzeczywisty rozkład zmiennej estymowanej.
Krzywa dla drzewa decyzyjnego jest najmniej stabilna i ma kilka punktów przegięcia i ekstremów lokalnych, co może wskazywać na lekkie przeuczenie modelu, jednak parametry dla niego zostały dobrane na zbiorze walidacyjnym tak, aby uzyskiwał on najlpesze wyniki. Z drugiej strony, krzywa dla lasu losowego jest już bardzo stabilna i ma małą wariancję w porównaniu do innych modeli (w szczególności drzewa decyzyjnego), co potwierdza teoretyczne własności tego modelu.

Poniższy wykres przedstawia krzywą frakcji danych, które zostały sklasyfikowane błędem bezwzględnym większym od wybranego przed standaryzacją zmiennej y, po nauczeniu każdego z modeli na całości danych treningowych. Dokładniej, dla ustalonego modelu, który dla $i-$tej obserwacji ze zbioru testowego przewiduje wartość $\widehat{y}^{(i)}$ oraz punktu x na poziomej osi wartość $f(x)$ wynosi

$$f(x) = \frac{1}{m_2} \sum_{i=1}^{m_2} \mathbf{1}[|y^{(i)} - \widehat{y}^{(i)}| > x]$$

gdzie m_2 jest liczbą obserwacji w zbiorze testowym.

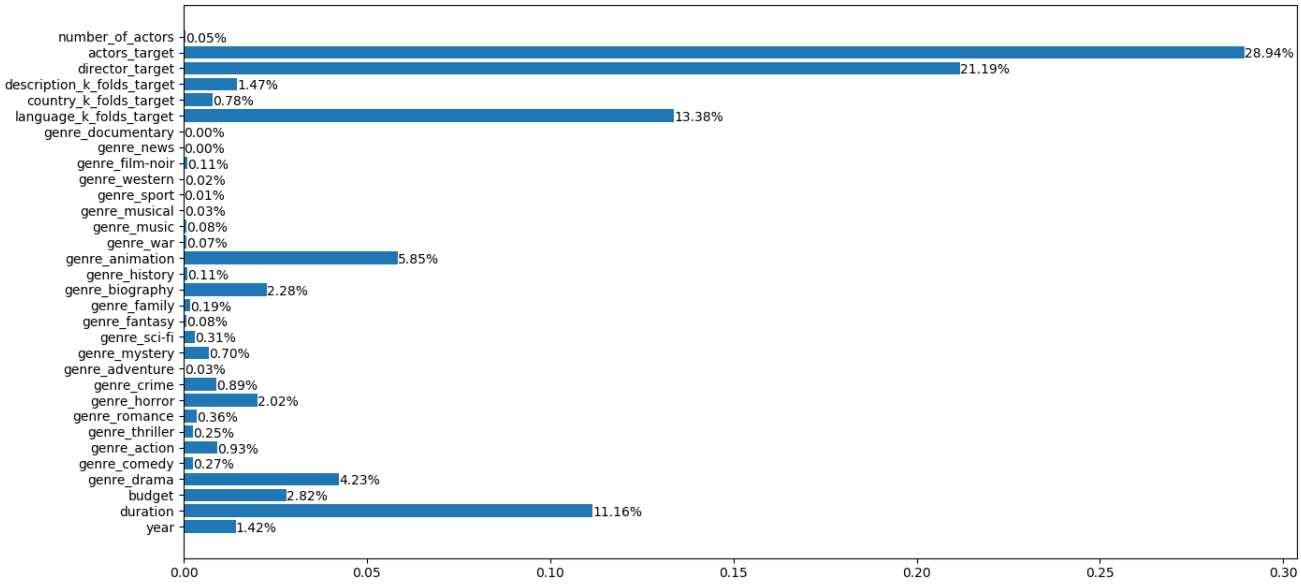


Rysunek 4: Frakcja obserwacji, dla których model estymował zmienną y z danym błędem bezwzględnym

Jak widać, wszystkie modele są bardzo porównywalne do siebie oraz dla połowy danych ze zbioru testowego każdy z modeli przewiduje końcową ocenę filmu z dokładnością ± 0.5 pkt. Ponadto, liczba obserwacji, na których model pomylił się znacznie (o więcej niż 2) jest prawie zerowa.

4 Wpływ poszczególnych cech na wynik

Wykres został przygotowany na podstawie modelu, który uzyskał najlepsze wyniki, czyli gradient boostingu ze stratą będącą funkcją hubera. W celu oceny ważności cech, zastosowano metodę 'permutation importance', polegającą na sprawdzeniu, jak zmienia się średni błąd kwadratowy modelu po losowej permutacji liczb w danej kolumnie macierzy danych testowych. Każdą wartość uśredniono na 100 losowych permutacjach każdej kolumny.



Rysunek 5: Ważność poszczególnych cech

5 Wnioski

- Wszystkie modele osiągają podobne wyniki, krzywe uczenia na danych treningowych i testowych zbiegają do bardzo bliskich sobie wartości, co wskazuje na to, że dalsza poprawa wyników może być trudna do uzyskania
- Poprawa hiperparametrów modelów lub zmiana modeli na inne raczej nie przyniesie znaczącej poprawy wyników. W celu osiągnięcia lepszych rezultatów można spróbować wyciągnąć więcej cech z bazy danych i inaczej je zakodować, jednak należy pamiętać, że ocena filmu wyłącznie na podstawie metadanych nie może być doskonała oraz fakt, że ludzie oceniają filmy w pewien sposób losowo.
- Kodowanie cech poprzez *target encoding* zachowuje się lepiej niż kodowanie zero jedynkowe i pozwala na uzyskanie informacji z cechy, która może przyjmować wiele wartości
- Opis filmu ma dość mały wpływ na ocenę filmu i prawdopodobnie można uzyskać jeszcze lepsze wyniki stosując lepszą analizę opisu, jednak z drugiej strony problemem jest to, że opis w bazie danych często nie jest podany w całości
- Budżet ma stosunkowo niski wpływ na ocenę filmu

Literatura

[1] Jerome H. Friedman *Greedy Function Approximation: A Gradient Boosting Machine*. Stanford University, 1999

[2] *Getting Deeper into Categorical Encodings for Machine Learning* towardsdatascience.com

[3] *Target encoding done the right way* maxhalford.github.io