Przewidywanie popularności artykułów

KWD

Szymon Nowak
Paulina Wyskiel

06.01.2023

Streszczenie

Głównym celem projektu jest wytrenowanie modeli pozwalających na predykcję popularności artykułów w sieciach społecznościowych w oparciu o zadanie regresji. Początkowo wykonana zostaje analiza danych oraz ich standaryzacja. Następnie problem ten jest badany wykorzystując regresję liniową wraz z innymi algorytmami pozwalającymi na ocenę i wybór najlepszego wyniku. Projekt zostaje rozszerzony o zadanie klasyfikacji ze względu na próbę znalezienia optymalnego rozwiązania.

Spis treści

Rozdział 1

Wprowadzenie

1.1 Opis problemu

Na podstawie zestawu danych Online News Popularity podsumowującego zestaw cech artykułów opublikowanych przez Mashable w okresie dwóch lat należy przewidzieć popularność artykułów w sieciach społecznościowych. Zestaw cech obejmuje zależności dotyczące tekstu, wystąpień słów kluczowych, ale również dodatkowej zawartości takiej jak zdjęcia oraz dni, w które artykuł się ukazał.

1.2 Opis danych

Analizowany zbiór danych składa się z **39797** próbek opisanych **61** cechami, z czego jedna z nich jest celem predykcji, a dwie należy odrzucić jako nie wpływające na jakość predykcji.

0. url: Adres URL artykułu (nieprognozująca)
1. timedelta: Ilość dni między opublikowaniem artykułu, a zebraniem statystyk (nieprognozująca)
2. n_tokens_title: Ilość słów w tytule
3. n_tokens_content: Ilość słów w artykule
4. n_unique_tokens: Stosunek unikatowych słów w artykule
5. n_non_stop_words: Stosunek słów nieodrzucanych przez wyszukiwarki
6. n_non_stop_unique_tokens: Stosunek unikatowych słów nieodrzucanych przez wyszukiwarki
7. num_hrefs: Ilość linków
8. num_self_hrefs: Ilość linków do innych artykułów Mashable

9. num_imgs: Ilość obrazów
10. num_videos: Ilość filmów
11. average_token_length: Średnia długość słów w artykule
12. num_keywords: Ilość słów kluczowych
13. data_channel_is_lifestyle: Czy kategoria to 'Lifestyle'?
14. data_channel_is_entertainment: Czy kategoria to 'Entertainment'?
15. data_channel_is_bus: Czy kategoria to 'Business'?
16. data_channel_is_socmed: Czy kategoria to 'Social Media'?
17. data_channel_is_tech: Czy kategoria to 'Tech'?
18. data_channel_is_world: Czy kategoria to 'World'?
19. kw_min_min: Najgorsze słowo kluczowe (min. shares)
20. kw_max_min: Najgorsze słowo kluczowe (max. shares)
21. kw_avg_min: Najgorsze słowo kluczowe (avg. shares)
22. kw_min_max: Najlepsze słowo kluczowe (min. shares)
23. kw_max_max: Najlepsze słowo kluczowe (max. shares)
24. kw_avg_max: Najlepsze słowo kluczowe (avg. shares)
25. kw_min_avg: Średnie słowo kluczowe (min. shares)
26. kw_max_avg: Średnie słowo kluczowe (max. shares)
27. kw_avg_avg: Średnie słowo kluczowe (avg. shares)
28. self_reference_min_shares: Minimalna liczba udostępnień artykułów Mashable do których odnosi się artykuł
29. self_reference_max_shares: Maksymalna liczba udostępnień artykułów Mashable do których odnosi się artykuł
30. self_reference_avg_sharess: Średnia liczba udostępnień artykułów Mashable do których odnosi się artykuł
31. weekday_is_monday: Czy artykuł opublikowano w Poniedziałek?
32. weekday_is_tuesday: Czy artykuł opublikowano we Wtorek?
33. weekday_is_wednesday: Czy artykuł opublikowano w Środek?
34. weekday_is_thursday: Czy artykuł opublikowano w Czwartek?
35. weekday_is_friday: Czy artykuł opublikowano w Piątek?
36. weekday_is_saturday: Czy artykuł opublikowano w Sobota?
37. weekday_is_sunday: Czy artykuł opublikowano w Niedziela?
38. is_weekend: Czy artykuł opublikowano w Weekend?
39. LDA_00: Bliskość do LDA tematu 0
40. LDA_01: Bliskość do LDA tematu 1
41. LDA_02: Bliskość do LDA tematu 2

42. LDA_03: Bliskość do LDA tematu 3
43. LDA_04: Bliskość do LDA tematu 4
44. global_subjectivity: Subiektywność tekstu
45. global_sentiment_polarity: Biegunowość sentymentalna tekstu
46. global_rate_positive_words: Ilość pozytywnych słów w artykule
47. global_rate_negative_words: Ilość negatywnych słów w artykule
48. rate_positive_words: Stosunek pozytywnych słów wśród nie neutralnych
49. rate_negative_words: Stosunek negatywnych słów wśród nie neutralnych
50. avg_positive_polarity: Średnia biegunowość pozytywnych słów
51. min_positive_polarity: Minimalna biegunowość pozytywnych słów
52. max_positive_polarity: Maksymalna biegunowość pozytywnych słów
53. avg_negative_polarity: Średnia biegunowość negatywnych słów
54. min_negative_polarity: Minimalna biegunowość negatywnych słów
55. max_negative_polarity: Maksymalna biegunowość negatywnych słów
56. title_subjectivity: Subiektywność tematu
57. title_sentiment_polarity: Biegunowość tematu
58. abs_title_subjectivity: Absolutna wartość subiektywności tematu
59. abs_title_sentiment_polarity: Absolutna wartość biegunowości tematu
60. shares: Ilość udostępnień (cel)

Rozdział 2

Opis metody

1.1 Wprowadzenie teoretyczne

Regresja liniowa

Regresja liniowa to wariant regresji w statystyce zakładający, że zależność pomiędzy zmienną objaśnianą a objaśniającą jest zależnością liniową. Jej celem jest wyznaczenie takich współczynników regresji, by pokazać i przewidzieć taką relację między dwiema zmiennymi lub czynnikami, która pozwoli na uzyskanie jak najmniejszego błędu szacowania. Zwykle, linia najlepszego dopasowania jest obliczana przy użyciu metody najmniejszych kwadratów. Uzyskana w ten sposób linia trendu pomaga w przewidzeniu, co będzie się działo w przyszłości.

Model regresji liniowej zakłada, że istnieje liniowa relacja między zmienną zależną y a wektorem regresorów x. Jego reprezentacje możemy zapisać jako:

$Y = X_i\beta + a, gdzie$
$Y - zmienna\ objaśniana$
$X_i - wartości\ predykatorów$
$\beta - współczynniki\ regresji\ dla\ poszczególnych\ predykatorów$
$a - wyraz\ wolny$

W postaci macierzowej opisujemy zbiór uczący $X = \{(x^i, y^i)\}, i = 1, \dots, m$ zakładając, że $x^{(i)} \epsilon \mathbb{R}^n, y^{(i)} \epsilon \mathbb{R}^z, gdzie\ jako\ z\ przyjmujemy\ 1.$

$$X = \begin{bmatrix} - & x^{1T} & - \\ - & x^{2T} & - \\ \dots & \vdots & \dots \\ - & x^{mT} & - \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ \dots & \vdots & & \dots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Regresja logistyczna

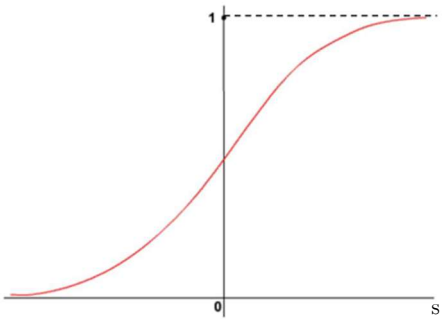
Regresja logistyczna jest modelem matematycznym pozwalającym na wykonanie zadania klasyfikacji. Polega na oszacowaniu prawdopodobieństwa wystąpienia zdarzenia na podstawie danego zestawu danych zmiennych niezależnych, które powinny być wybrane tak, aby istotnie wpływały na zmienną Y.

Ogólny model logistyczny przyjmuje postać:

$P\ (Y{=}1|X_1{=}x_1,\ldots,X_k{=}x_k){=}\pi(x_1,\ldots,x_k){=}1{-}P(Y{=}0|X_1{=}x_1,\ldots,X_k{=}x_k)$, gdzie
$(X_1, \ldots, X_k)$ - wektor zmiennych objaśniających
Y - binarna zmienną objaśniana

Model regresji logistycznej, w której opisujemy wpływ kilku zmiennych X_1, X_2, ..., X_k na dychotomiczną zmienną Y, oparty jest o funkcję logistyczną, która ma postać $g(s) = \frac{1}{1+e^{-\beta s}}, gdzie\ s = \theta^T x.$

Funkcja sigmoidalna nigdy nie osiąga 0 ani 1.



Klasyfikacja

Klasyfikacja w uczeniu maszynowym to podejście uczenia nadzorowanego, w którym na podstawie zestawu danych dokonywane jest kategoryzowanie na klasy. Może być przeprowadzane dla danych ustrukturyzowanych jak i

nieustrukturyzowanych. Głównym celem klasyfikacyjnego modelowania predykcyjnego jest określenie, do której klasy będą należeć nowe dane.

Metoda K-najbliższych sąsiadów (KNN)

Metoda K-najbliższych sąsiadów (ang. K-Nearest Neighbours) jest parametryzowanym, nadzorowanym klasyfikatorem stosowanym w statystyce w celu przewidywania grupowania poszczególnych punktów danych. Pozwala na przydzielenie klasy, do której należy większość z jego k-sąsiadów, oznacza to, że przyznawana jest etykieta, która jest najczęściej reprezentowana wokół danego punktu danych. Algorytm KNN należy do grupy modeli „leniwego uczenia się", czyli nie jest trenowany do generowania prognoz, a przewiduje on na podstawie k najbliższych obserwacji i ich etykiet jaki wynik należy przypisać analizowanej próbce.

Działanie algorytmu:

Zakładamy, że istnieje zbiór uczący X zawierający obserwacje C, z których każda ma przypisany wektor zmiennych objaśniających $X_1 ... X_n$, dla której prognozujemy wartość zmiennej objaśnianej Y.

Wartości zmiennych objaśniających dla danego C są porównywane z wartościami tych zmiennych dla każdego C w X, a uzyskanie prognozy polega na otrzymaniu uśrednionej zmiennej objaśnianej Y dla wybranych obserwacji C.

Drzewo decyzyjne

Drzewo decyzyjne to algorytm uczenia nadzorowanego, opierający się na rekurencyjnym podejściu dziel i zwyciężaj. Wykorzystywany jest zarówno do zadań klasyfikacji jak i regresji. Jego struktura jest hierarchiczna i składa się na nią korzeń, gałęzie, węzły wewnętrzne, inaczej decyzyjne i liście, które reprezentują wszystkie możliwe wyniki w zbiorze danych. Celem wykorzystania drzewa decyzyjnego jest utworzenie modelu, który można użyć do przewidywania klasy poprzez naukę reguł decyzyjnych uzyskanych z danych szkoleniowych. Przewidywanie etykiety klasy rozpoczyna się od korzenia drzewa, a następnie poprzez porównania podążamy za gałęzią, która odpowiada uzyskanej wartości, przeskakujemy do kolejnego węzła. Wadą

drzew jest możliwa ich niestabilność, ponieważ nawet prosta zmiana danych może zaburzyć całą jego strukturę.

SGD – Stochastyczny Spadek Gradientu

Stochastyczny Spadek Gradientu to algorytm optymalizacji pozwalający na znajdowanie optymalnych rozwiązań. W SDG do wykonania każdej iteracji wykorzystuje się jedną próbkę z zestawu danych treningowych. Następnie dane są tasowane, aby zapobiec cyklom, a gradient główny jest aproksymowany przez gradient w pojedynczej próbie, aż do uzyskania przybliżonego minimum. Wadą tego rozwiązania jest potrzeba wielu hiperparametrów i wrażliwość na skalowanie funkcji.

Regresja Lasso

Lasso (ang. Least Absolute Shrinage and Selection Operator – operator najmniejszej bezwzględnej redukcji i wyboru) to metoda analizy modelu regresji liniowej. Przeprowadza regularyzację L1 oraz selekcję atrybutów, aby zwiększyć dokładność predykcji. Podczas regularyzacji R1 poszczególne cechy sprowadzane są do 0.

Błąd średniokwadratowy

Błąd średniokwadratowy (ang. Mean Squared Error) to wartość oczekiwana kwadratu błędu, czyli różnicy pomiędzy wartością uzyskaną za pomocą algorytmu a wartością rzeczywistą. Jeśli błąd średniokwadratowy jest mały to model jest bardziej dopasowany.

Walidacja krzyżowa

Walidacja krzyżowa (cross-validation) służy do określenia jakości modelu w trakcie uczenia. Jej działanie polega na podziale zbioru danych na podzbiory i utworzeniu grupy uczącej i grupy testowej.

GridSearchCV

Jest procesem dostrajania hiperparametrów w celu określenia optymalnych wartości dla danego modelu. Jego działanie polega na testowaniu kombinacji wartości przekazanych w słowniku i ocenienia modelu dla tych kombinacji wykorzystując metodę walidacji krzyżowej.

Współczynnik determinacji R^2

Jest miarą statystyczną jakości dopasowania modelu uczącego do zmiennej. Jego wartość wyznaczana jest jako średnia arytmetyczna kwadratów odchyleń wartości cechy od średniej arytmetycznej dla tej cechy. Pozwala na uzyskanie informacji o tym jak bardzo zmiany wartości są zdeterminowane zmianami w zakresie innej cechy.

1.2 Badania symulacyjne

Analiza eksploracyjna danych

Pierwszym krokiem jaki należy podjąć rozważając zadanie jakim jest przewidywanie popularności artykułu zawartego na stronie internetowej, jest przyjrzenie się dostępnym wartościom w poszukiwaniu różnego rodzaju niechcianych artefaktów bądź zjawisk.

Przejrzenie dostępnych cech w zbiorze, którym dysponujemy umożliwia wywnioskowanie, które spośród nich wpływają na popularność artykułu, a jakie są nią niezwiązane. Możliwe jest jednoznaczne stwierdzenie, że dwie spośród sześćdziesięciu cech są niezwiązane z celem, czyli ilością udostępnień oznaczoną etykietą *„shares”*. Są to kolumny *„url”* oraz *„timedelta”,* będące kolejno odnośnikiem do artykułu oraz różnicą czasu między publikacją treści, a zebraniem statystyk jej dotyczących. W wyniku tego stwierdzenia ze zbioru danych cechy te zostały odrzucone w wyniku czego uzyskano obiekt DataFrame składający się z 59 kolumn oraz 39644 wierszy.

Kolejnym krokiem podczas analizy danych jest przeszukanie zbioru pod kątem brakujących wartości. W przypadku odnalezienia takich pól, należałoby dobrać odpowiednią wartość zastępującą braki, w celu poprawienia jakości modelu uczenia maszynowego. Wykorzystywany zbiór nie posiada jednak wartości ***null***, dlatego wstawianie sztucznych wartości nie jest konieczne.

Potwierdzenie powyższego stwierdzenia zawiera się w wydruku metody *info()*, wykonanej na analizowanym zbiorze.

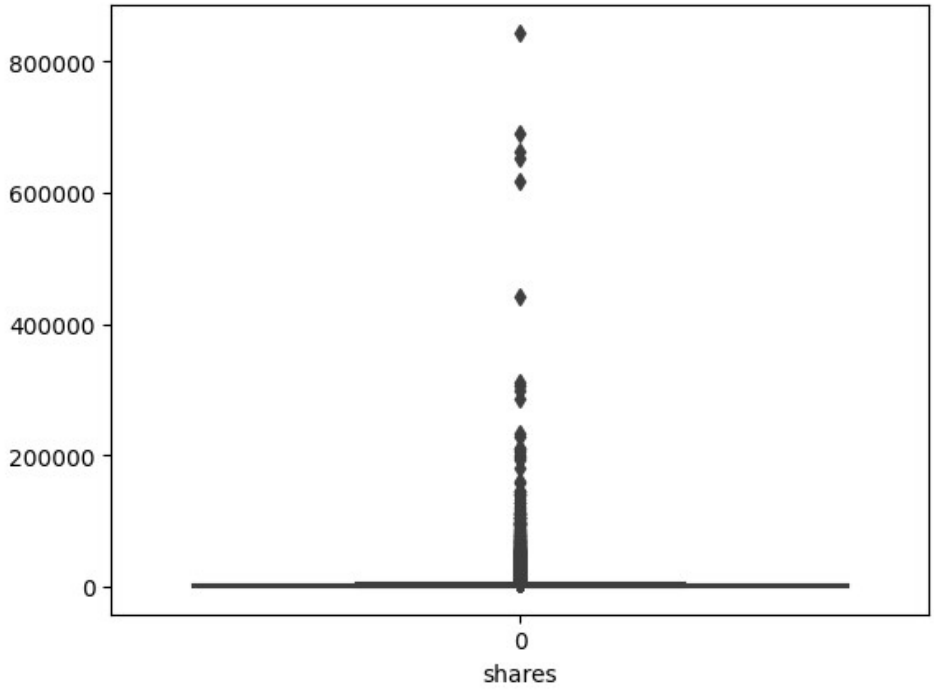
Nr	**Nazwa cechy**	**Ilość**		**Typ**
0	n_tokens_title	39644	non-null	float64
1	n_tokens_content	39644	non-null	float64
2	n_unique_tokens	39644	non-null	float64
3	n_non_stop_words	39644	non-null	float64
4	n_non_stop_unique_tokens	39644	non-null	float64
5	num_hrefs	39644	non-null	float64
6	num_self_hrefs	39644	non-null	float64
7	num_imgs	39644	non-null	float64
8	num_videos	39644	non-null	float64
9	average_token_length	39644	non-null	float64
10	num_keywords	39644	non-null	float64
11	data_channel_is_lifestyle	39644	non-null	float64
12	data_channel_is_entertainment	39644	non-null	float64
13	data_channel_is_bus	39644	non-null	float64
14	data_channel_is_socmed	39644	non-null	float64
15	data_channel_is_tech	39644	non-null	float64
16	data_channel_is_world	39644	non-null	float64
17	kw_min_min	39644	non-null	float64
18	kw_max_min	39644	non-null	float64
19	kw_avg_min	39644	non-null	float64
20	kw_min_max	39644	non-null	float64
21	kw_max_max	39644	non-null	float64

22	kw_avg_max	39644	non-null	float64
23	kw_min_avg	39644	non-null	float64
24	kw_max_avg	39644	non-null	float64
25	kw_avg_avg	39644	non-null	float64
26	self_reference_min_shares	39644	non-null	float64
27	self_reference_max_shares	39644	non-null	float64
28	self_reference_avg_sharess	39644	non-null	float64
29	weekday_is_monday	39644	non-null	float64
30	weekday_is_tuesday	39644	non-null	float64
31	weekday_is_wednesday	39644	non-null	float64
32	weekday_is_thursday	39644	non-null	float64
33	weekday_is_friday	39644	non-null	float64
34	weekday_is_saturday	39644	non-null	float64
35	weekday_is_sunday	39644	non-null	float64
36	is_weekend	39644	non-null	float64
37	LDA_00	39644	non-null	float64
38	LDA_01	39644	non-null	float64
39	LDA_02	39644	non-null	float64
40	LDA_03	39644	non-null	float64
41	LDA_04	39644	non-null	float64
42	global_subjectivity	39644	non-null	float64
43	global_sentiment_polarity	39644	non-null	float64
44	global_rate_positive_words	39644	non-null	float64
45	global_rate_negative_words	39644	non-null	float64

46	rate_positive_words	39644	non-null	float64
47	rate_negative_words	39644	non-null	float64
48	avg_positive_polarity	39644	non-null	float64
49	min_positive_polarity	39644	non-null	float64
50	max_positive_polarity	39644	non-null	float64
51	avg_negative_polarity	39644	non-null	float64
52	min_negative_polarity	39644	non-null	float64
53	max_negative_polarity	39644	non-null	float64
54	title_subjectivity	39644	non-null	float64
55	title_sentiment_polarity	39644	non-null	float64
56	abs_title_subjectivity	39644	non-null	float64
57	abs_title_sentiment_polarity	39644	non-null	float64
58	shares	39644	non-null	int64

Wykreślenie wykresów pudełkowych dla każdej z cech pozwala przyjrzeć się rozkładowi danych w każdej z nich. Większość z kolumn nie zawiera wielu outlinerów lub ta dysproporcja jest niewielka. Inna sytuacja dotyczy cechy będącej celem analizowanego problemu, gdzie część z wartości znacznie odstaje od reszty, wprowadzając niechciane zaszumienie.

Fakt ten obrazuje poniższy wykres:



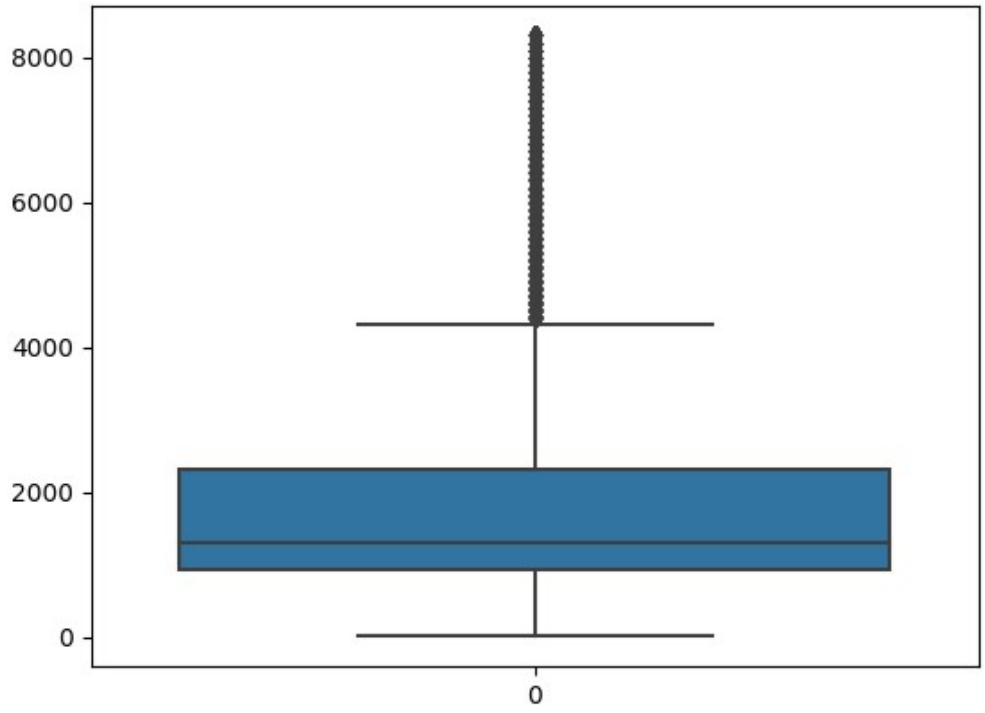
Rozwiązaniem problemu wartości odstających jest pozbycie się ich. Odrzucone zostały wszystkie próbki, dla których wartość *„shares”* przekraczała granicę wartości *ekstremalnie odstających*, wyznaczonych poprzez dodanie do trzeciego kwartyla wartości rozstępu międzykwartylowego pomnożonego przez 3. Sposób ten wyznacza wartość graniczną równą *8362*, powyżej tej liczby znajduje się *2765* próbek, które zostają odrzucone.

Wzory:

```
Q1 = df['shares'].quantile(q=0.25)
Q3 = df['shares'].quantile(q=0.75)

IQR = Q3-Q1
extreme_outliers_bound = Q3+(IQR*3)
```

Wykres cechy *shares* po odcięciu wartości odstających:



Danym również warto się przyjrzeć pod kątem opisujących je wartości, algorytmy uczenia maszynowego mają tendencję do lepszego funkcjonowania przy wykorzystaniu danych znormalizowanych bądź ustandaryzowanych. Przedstawiony zbiór zawiera wartości z różnych przedziałów, dlatego należy go poddać procesowi standaryzacji.

Wartość średnia oraz odchylanie standardowe każdej z cech przed standaryzacją:

```
---Mean---
[ 1.03976518e+01  5.48003878e+02  1.90081076e-02  2.82545622e-02
  1.76252068e-02  1.06974159e+01  3.30171642e+00  4.39239133e+00
  1.20724532e+00  4.00848721e+00  7.20968573e+00  5.22519591e-02
  1.78855175e-01  1.62639985e-01  5.80818352e-02  1.87993167e-01
  2.19420266e-01  2.61909759e+01  1.01051951e+03  3.04242604e+02
  2.90515594e+03 -8.14325096e+03  5.85250522e+02  1.09907408e+03
  5.10349700e+03  3.07230023e+03  2.18519700e+03  4.04851222e+03
  3.35761143e+03  1.67141191e-01  1.86772960e-01  1.88779522e-01
  1.84169853e-01  1.44174191e-01  6.09832154e-02  6.79790667e-02
  1.28962282e-01  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  1.08462811e-04  0.00000000e+00
  0.00000000e+00  0.00000000e+00  3.50606036e-02  7.86355378e-04
  1.89809919e-04  1.89809919e-04  3.69831069e-01 -1.30155373e-03
 -1.37910464e-01 -1.30155373e-03  6.11459096e-02  8.64990916e-03
  0.00000000e+00  1.70015456e-02]
--std---
[2.11162144e+00 4.66661104e+02 3.65024960e+00 5.42590596e+00
 3.38468222e+00 1.10487562e+01 3.85800955e+00 8.14860139e+00
 3.99562189e+00 7.62084926e-01 1.91427708e+00 2.22534698e-01
 3.83231003e-01 3.69036882e-01 2.33898131e-01 3.90706714e-01
 4.13853855e-01 6.97236848e+01 2.00868310e+03 5.28883479e+02
 8.86347807e+03 1.21342572e+04 1.90284238e+04 1.12103329e+03
 3.44221391e+03 1.24284316e+03 4.74607846e+03 7.89676494e+03
 6.18559309e+03 3.73101881e-01 3.89729164e-01 3.91333380e-01
 3.87622649e-01 3.51266272e-01 2.39299525e-01 2.51709978e-01
 3.35158189e-01 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 1.04139832e-02 0.00000000e+00
 0.00000000e+00 0.00000000e+00 1.83933025e-01 2.80310011e-02
 1.37758445e-02 1.37758445e-02 4.82758790e-01 3.60535669e-02
 3.44805986e-01 3.60535669e-02 2.39597762e-01 1.30102747e-01
 0.00000000e+00 1.29276808e-01]
```

Wartość średnia oraz odchylanie standardowe każdej z cech po standaryzacji:

```
---Mean---
[ 3.08269849e-16 -2.42762506e-17 -9.63343279e-19  1.73401790e-18
  2.50469253e-18  2.08082148e-17  9.24809548e-18  3.08269849e-17
  6.35806564e-18  1.47969528e-16  1.78796513e-16  5.39472236e-17
 -1.73401790e-17  1.84961910e-17 -4.00750804e-17  5.24058744e-17
 -3.69923819e-17  1.78796513e-16 -2.62029372e-17  0.00000000e+00
  2.62029372e-17 -5.93419460e-17  2.23495641e-17  0.00000000e+00
 -4.40729550e-17 -1.72631116e-16 -2.31202387e-18  1.38721432e-17
  3.39096834e-17  9.78756772e-17 -2.00375402e-17 -1.34868059e-17
  9.71050025e-17 -2.89002984e-17  2.87076297e-17 -1.86888596e-17
 -2.27349014e-17  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  2.31202387e-18  0.00000000e+00
  0.00000000e+00  0.00000000e+00  2.92856357e-17  2.31202387e-17
 -3.85337312e-19 -3.85337312e-19  3.69923819e-17 -1.46428178e-17
  8.32328593e-17 -1.46428178e-17 -1.15601193e-18 -3.81483939e-17
  0.00000000e+00 -4.08457550e-17]
--std---
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.]
```

Spreparowane w ten sposób dane można podzielić na zbiory uczące i treningowy oraz wykorzystać w dalszym etapie rozwiązywania problemu.

Regresja liniowa

W celu odnalezienia najlepszego modelu konieczne jest przeanalizowanie oraz porównanie wydajności kilku algorytmów uczenia maszynowego. W celu rozwiązania rozważanego problemu wykorzystano algorytmy:

- Regresji liniowej
- K-najbliższych sąsiadów
- Drzewa decyzyjnego
- SGD – Stochastycznego Spadku Gradientu

Decyzję o wyborze najlepszego modelu do wykonywanego zadania można oprzeć na wynikach walidacji krzyżowej. Algorytmy osiągające lepszą wydajność w sprawdzianie krzyżowym, prawdopodobnie będą sprawdzać się lepiej podczas realizacji prawdziwych zadań, dlatego w procesie strojenia

hiperparametrów najlepiej się skupić na nich. Powyżej wymienione modele osiągnęły następujące wartości:

Wyniki walidacji krzyżowej		
Model	Średnia	Odchylenie standardowe
Regresja liniowa	-5.758438120515409e+23	1.0519881399385761e+24
Regresja SGD	-1000701832453976.4	824348615900667.5
Drzewo decyzyjne	-0.8284908362111827	0.035106021792625855
K-najbliższych sąsiadów	-0.05086886167192697	0.010682403590046324

Otrzymane dane pozwalają wyznaczyć najlepiej rokujący model. Jest nim model k-najbliższych sąsiadów dla którego średnia wartość współczynnika determinacji R^2 wynosi w przybliżeniu *-0.05,* przy odchyleniu standardowym zaokrąglanym do *0.01.* Algorytm ten przyjmuje kilka parametrów wpływających na proces uczenia, ich odpowiednie dobranie pozwala na poprawienie jakości predykcji. Proces selekcji parametrów wykonany został z użyciem metody przeszukiwania siatki – obiekt GridSearchCV sprawdził wszystkie możliwe kombinacje dostarczonych parametrów trenując z ich wykorzystaniem modele oraz dokonując na nich sprawdzianu krzyżowego. Sprawdzone zostało **80** kombinacji, osiągając najlepszy wynik równy **0.0698583573017971**.

Przeszukiwanie wykonane zostało na wypisanych poniżej parametrach (wartości **pogrubione** oznaczają parametry wybrane w procesie przeszukiwania siatki)**:**

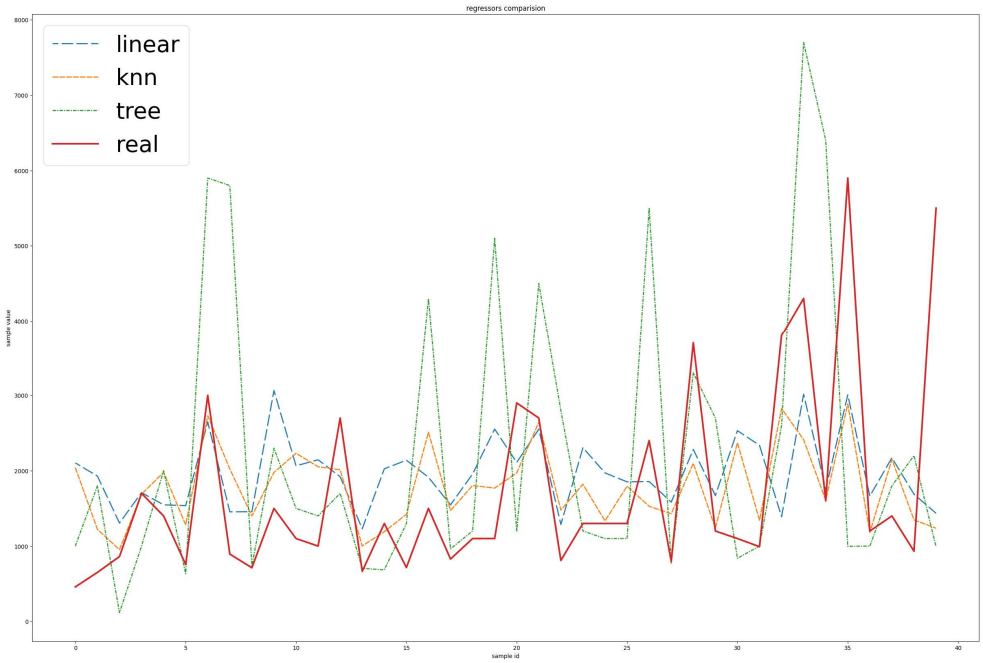
- Leaf_size: **5**, 15, 30, 45, 60
- Weights: uniform, **distance**
- N_neighbours: 5, 10, 15, **20**
- P: **1**, 2

W celach porównawczych wytrenowane oraz ocenione zostały wszystkie z analizowanych modeli dając bardzo różniące się od siebie wyniki. Podczas

ich analizy warto zwrócić uwagę na fakt, że model regresji liniowej sprawdził się lepiej dla danych testowych, pomimo osiągnięcia znacznie gorszych wyników podczas walidacji krzyżowej. Wyraźna jest również różnica zarówno w błędzie średniokwadratowym jak i współczynnikowi korelacji dla SGD w porównaniu do reszty modeli, wynik dla tego algorytmu jest o kilka rzędów gorszy od innych.

Ocena predykcji		
Model	Błąd średniokwadratowy	Współczynnik R^2
Regresja liniowa	2154716.20	0.09
Regresja SGD	9939694596152604753 92.00	-420592445521455.44
K-najbliższych sąsiadów	2216401.44	0.06
Drzewo decyzyjne	4236816.98	-0.79

Mimo niezadowalających wyników oceny, przyglądając się części próbek możemy stwierdzić, że mimo wszystko przewidywane wyniki nie odbiegają tak bardzo od rzeczywistości. Punkty oznaczające wartości rzeczywiste oraz otrzymane jako predykcje często znajdują się blisko siebie, jednak wartości przewidywanej cechy liczone są setkach, bądź tysiącach co powoduje wysokie wartości błędów. Charakterystyka samej cechy również jest ciężka do przewidzenia oraz zależna od wielu czynników niedostępnych w analizowanym zbiorze, dlatego zbudowanie dokładnego modelu dla przedstawionego zadania regresji jest niemożliwe.



Sposobem na poprawę jakości predykcji mogłoby być stworzenie bardziej złożonego modelu składającego się z wielomianowych kombinacji cech. Podjęta została próba wytrenowania modeli posługującymi się kombinacjami wielomianu stopnia drugiego, wyniki jednak nie poprawiły się, natomiast wybranie cech najbardziej wpływających na wynik za pomocą algorytmu RFE nie było możliwe, ze względu na bardzo długi czas trwania wykonywania operacji.

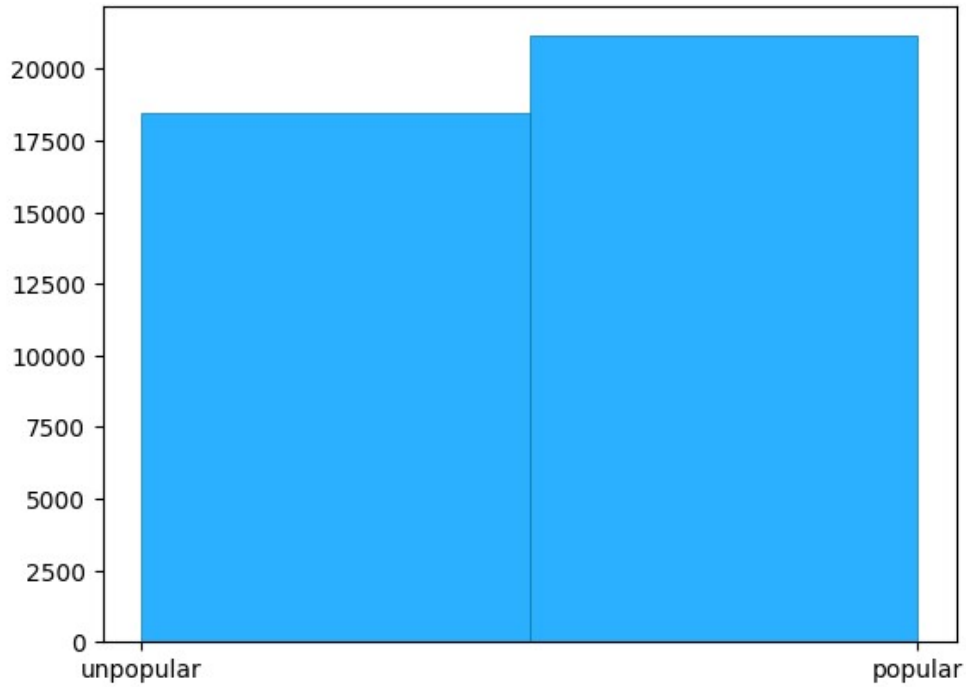
Ocena predykcji			
Zbiór danych	Model	Błąd średniokwadratowy	Współczynnik R^2
Kombinacje wielomianowe	Regresja liniowa	2228718.91	0.06
	Regresja SGD	4359514677089691490412341341067523777915146521804800 0.00	-184470349826801158234410825785719139101394862080 0.00
	Drzewo decyzyjne	4605168.43	-0.95
	K-najbliższych sąsiadów	2309584.37	0.02
Znormalizowane kombinacje wielomianowe	Regresja liniowa	2477577671.96	-1047.37
	Regresja SGD	2298670.39	0.03
	Drzewo decyzyjne	4604387.44	-0.95
	K-najbliższych sąsiadów	2335957.21	0.01
Ustandaryzowane kombinacje wielomianowe	Regresja liniowa	3465453055739375743817639526400 0	-1466386478431042420080640 0
	Regresja SGD	3338705616560394275389440 0	-141275403037926384.00
	Drzewo decyzyjne	6801560.43	-1.88
	K-najbliższych sąsiadów	2251317.28	0.05

Klasyfikacja

W celu odnalezienia optymalnego rozwiązania problemu rozważono również traktowanie problemu jako zadanie klasyfikacji oraz nieprzewidywanie wartości należącej do zbioru liczb rzeczywistych, a etykiety klasyfikującej artykuł jako popularny albo niepopularny.

Proces obróbki danych jest bardzo podobny do wykonanego w trakcie realizacji zadania regresji liniowej, tym razem jednak nie odrzucono wartości odstających. Kolumnę *shares* zastąpiono kolumną *popularity*, posiadającą wartość 1, jeśli próbka posiadała wartość *shares* większą od mediany wartości tej cechy.

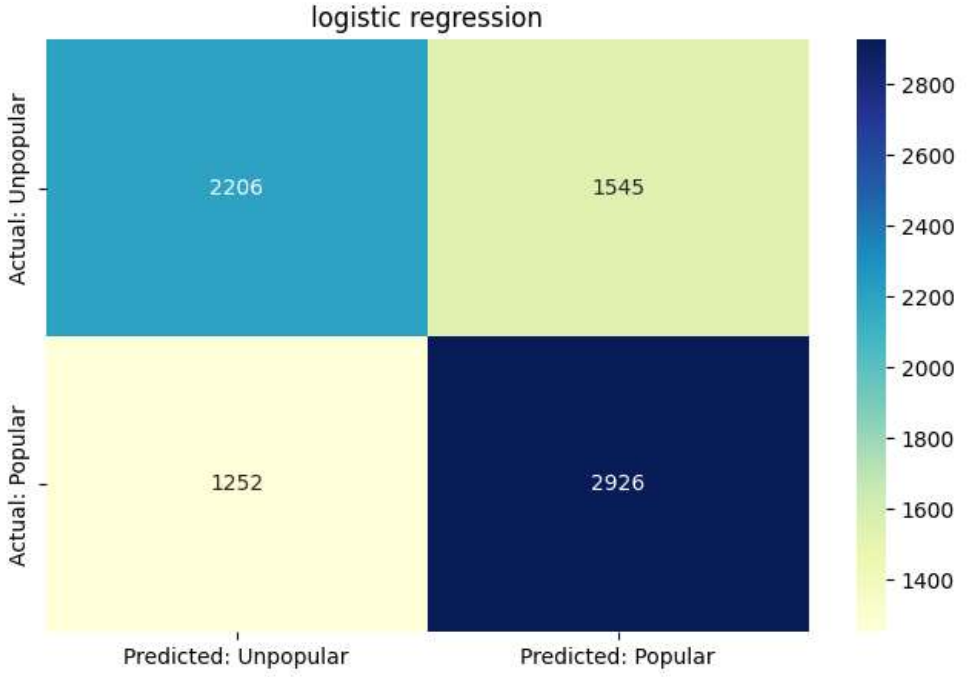
Rozkład popularności artykułów:

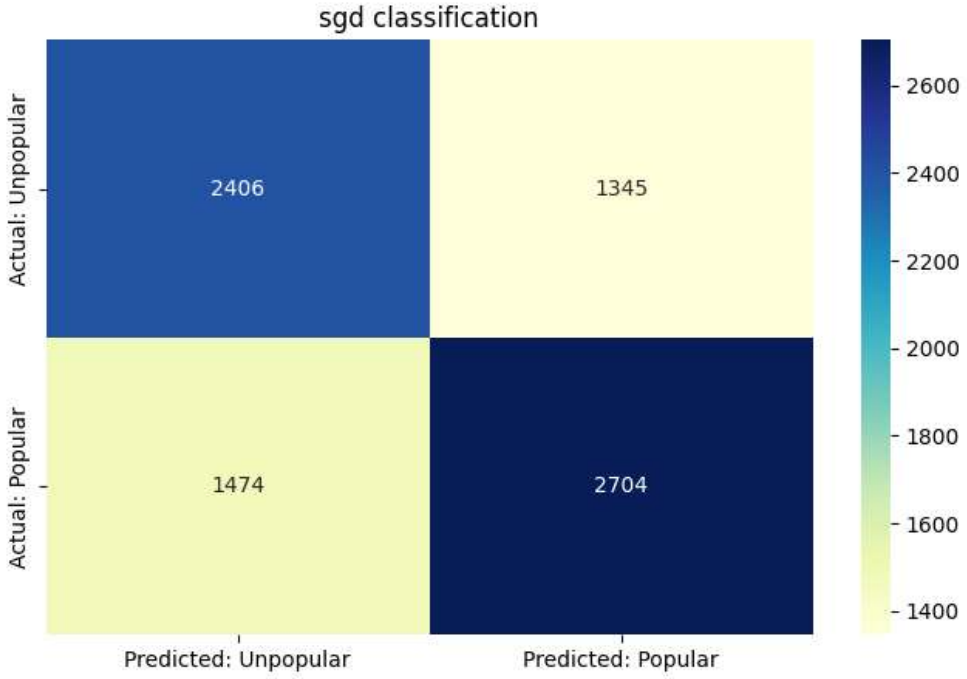


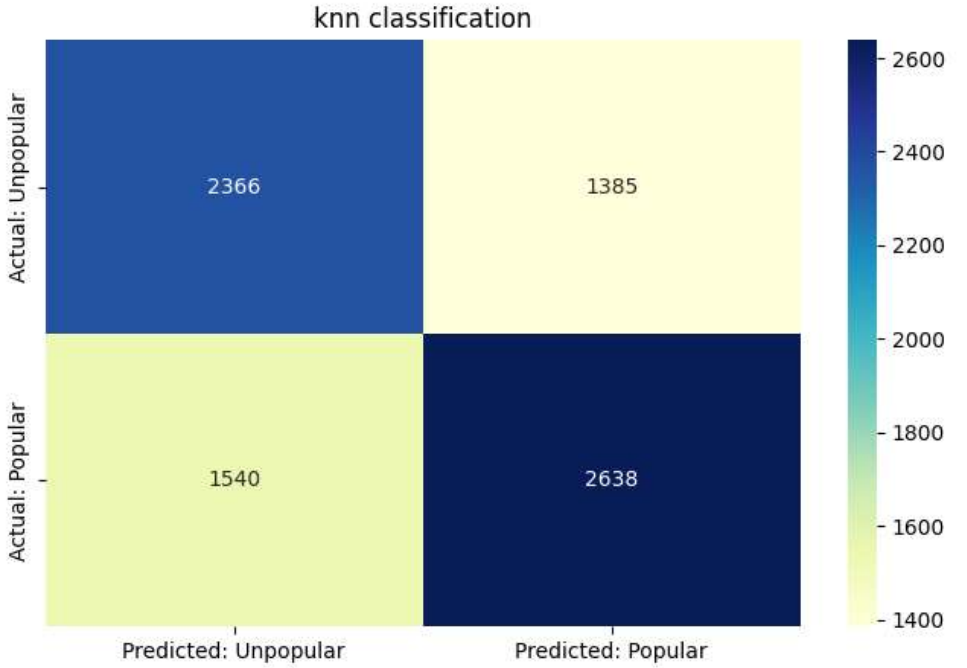
Wykorzystano takie same modele klasyfikacji jak w przypadku zadania regresji liniowej co pozwoliło uzyskać dokładność modeli oscylujących wokół wartości 0.6, dokładne wyniki zawiera tabela poniżej.

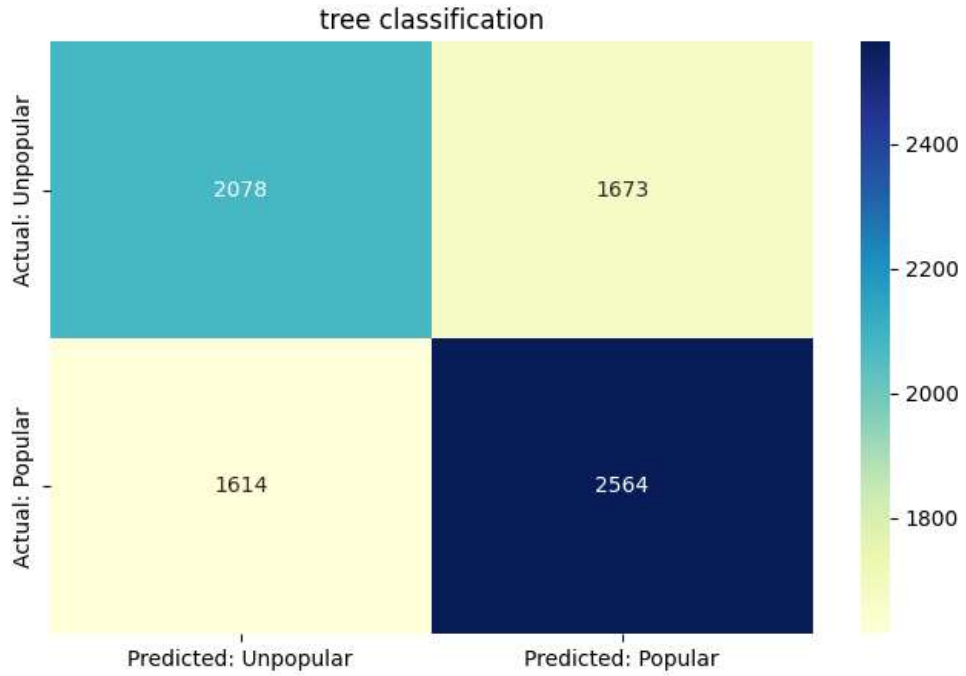
Klasyfikator	Dokładność
Regresja logistyczna	0.65
SGD	0.63
Drzewo decyzyjne	0.63
N-najbliższych sąsiadów	0.58

Macierze konfuzji analizowanych modeli:




sgd classification
Actual: Unpopular
Actual: Popular
2406
1345
1474
2704
Predicted: Unpopular
Predicted: Popular
2600
2400
2200
2000
1800
1600
1400


knn classification
Actual: Unpopular
Actual: Popular
2366
1385
1540
2638
Predicted: Unpopular
Predicted: Popular
2600
2400
2200
2000
1800
1600
1400



Analiza wyników predykcji wytrenowanych modeli pozwala wysnuć wnioski odnośnie ich wydajności, każdy z nich cechuje się bardzo podobną skutecznością osiągając w najgorszym przypadku dokładność **0.58** dla modelu drzewa decyzyjnego oraz najlepszym **0.65** dla regresji logistycznej. Pomimo poprawnej klasyfikacji większości z przypadków, modeli wciąż nie można nazwać dobrymi. Ich niską jakość obrazują powyższe wykresy przedstawiające macierze konfuzji, z których można odczytać wszystkie przypadki dobrze i źle zaklasyfikowane, łatwo zauważalna jest również bardzo duża ilość przypadków fałszywie pozytywnych oraz negatywnych w każdym z przypadków

Rodział 3

Podsumowanie

Osiągnięcie celu jakim było uzyskanie modelu pozwalającego na rozważanie problemu predykcji popularności artykułów online, okazało się nie być trywialne przy użyciu dostępnych danych. Są one niewystarczające, aby utworzyć model o satysfakcjonującej dokładności. Niezależnie od zastosowanego algorytmu uczenia maszynowego zarówno rozważanie problemu jako regresji jak i klasyfikacji dało niezadowalające rezultaty. Biorąc pod uwagę uzyskane wyniki, możliwe jest wysnucie następujących dwóch wniosków: dostępne dane posiadają zbyt mało szczegółów lub sam problem posiada zbyt złożoną bądź losową naturę, aby dokładnie przewidzieć jak popularny będzie analizowany wpis. Rozwiązaniem pierwszego z tych zagadnień mogłoby być zbieranie większej ilości statystyk, bądź innych danych. Natomiast niemożliwym jest pozbycie się pewnego elementu losowości jakim jest zainteresowanie wśród ludzi daną treścią, na które wpływa wiele czynników trudnych do zmierzenia.

Dodatek A

Kod programu – Regresja

```
%matplotlib inline

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import Normalizer
from sklearn.feature_selection import RFE
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split

df = pd.read_csv('OnlineNewsPopularity.csv', header=0)
names = df.columns
stripped = [s.strip() for s in names]
df.columns=stripped


# Analiza eksploracyjna danych
```

```
df.head()

# Odrzucanie url i timedelta jako nie prognozujące
df.drop(columns=['url', 'timedelta'],inplace=True)
df.describe()


# Sprawdzenie ilości danych NULL w każdej z cech.

df.info()


for i in df.columns:
    sns.boxplot(df[i])
    plt.xlabel(i)
    plt.show()


# Z analizy powyższych wykresów wynika, że większość z cech nie posiada
wielu danych odstających. Wyjątkiem jest cecha będąca naszym celem, czyli
shares, gdzie widać znaczy rozstrzał między wartościami.

data = df.iloc[:,:-1]
target = df.iloc[:,-1]
print(target.shape)


# Wykres pudełkowy celu.

sns.boxplot(target)
plt.xlabel("shares")
plt.show()


target.max()
```

```
plt.hist(target, bins=np.arange(target.min(), target.max()+1, 10_000))


# Po analizie histogramu oraz wykresu pudełkowego widoczne stają się
różnice między wartościami w danych, aby zminimalizować problem gene-
rowany przez dane odstające zostaną one odrzucone.

Q1 = df['shares'].quantile(q=0.25)
Q3 = df['shares'].quantile(q=0.75)

IQR = Q3-Q1
# pierwszy kwartyl
print("Pierwszy kwartyl: ", Q1)
# drugi kwartyl
print("Trzeci kwartyl: ", Q3)
# rozstęp międzykwartyly
print("Rozstęp międzykwartylny: ", IQR)
extreme_outliers_bound = Q3+(IQR*3)
print("Granica odcinająca ekstremalnie odstające wartości:
",extreme_outliers_bound)


print("Wielkość macierzy przed odcięciem wartości odstających: ", df.shape)
df_old = df
df.drop(df[df.shares > extreme_outliers_bound].index, inplace=True)
print("Wielkość macierzy po odcięciu wartości odstających: ", df.shape)


df.describe()


data = df.iloc[:,:-1]
target = df.iloc[:,-1]
print(target.shape)
print(target.loc[:10])
```

```
sns.boxplot(target)


data.hist(bins=100, figsize=(100,100))
plt.show()


data_np = np.array(data, dtype=np.int16)
target_np = np.array(target, dtype=np.int16)


print("First article in database")
print(data_np[1,:])
print('---Mean---')
print(data_np.mean(axis=0))
print('--std---')
print(data_np.std(axis=0))


scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_np)


print("First article in database")
print(scaled_data[1,:])
print('---Mean---')
print(scaled_data.mean(axis=0))
print('--std---')
print(scaled_data.std(axis=0))


news_train_data, news_test_data, \
news_train_target, news_test_target = \
train_test_split(scaled_data,target, test_size=0.2, random_state=10)


print("Training dataset:")
```

```
print("news_train_data:", news_train_data.shape)
print("news_train_target:", news_train_target.shape)

print("Testing dataset:")
print("news_test_data:", news_test_data.shape)
print("news_test_target:", news_test_target.shape)


def printCrossValScore(name, sc):
    print("Cross Validations scores for ", name)
    print("Wyniki: ", sc)
    print("Średnia: ", sc.mean())
    print("Odchylenie standardowe: ", sc.std())


from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
scores = cross_val_score(lin_reg, news_train_data, news_train_target,
scoring='r2', cv=5)
print(scores)


printCrossValScore("Linear Regression", scores)


sgd_reg = SGDRegressor()
scores = cross_val_score(sgd_reg, news_train_data, news_train_target,
scoring='r2', cv=5)
print(scores)


printCrossValScore("SGD Regression", scores)


tree_reg = DecisionTreeRegressor()
```

```
scores = cross_val_score(tree_reg, news_train_data, news_train_target,
scoring='r2', cv=5)
print(scores)


printCrossValScore("Decision Tree Regression", scores)


knn_reg = KNeighborsRegressor()
scores = cross_val_score(knn_reg, news_train_data, news_train_target,
scoring='r2', cv=5)
print(scores)


printCrossValScore("KNN Regression", scores)


param_grid = {
    'n_neighbors': [5,10,15,20], 'weights':['uniform','distance'],
'leaf_size':[5,15,30,45,60], 'p':[1,2]
}


grid_search = GridSearchCV(KNeighborsRegressor(), param_grid, scor-
ing="r2", cv=5, return_train_score=True, verbose=3, n_jobs=-3)
grid_search.fit(news_train_data, news_train_target)


print(grid_search.best_score_)
print(grid_search.best_params_)
print(grid_search.best_estimator_)


lin_reg.fit(news_train_data, news_train_target)
sgd_reg.fit(news_train_data, news_train_target)
tree_reg.fit(news_train_data, news_train_target)
knn_reg = grid_search.best_estimator_
```

```
knn_reg.fit(news_train_data, news_train_target)


lin_pred = lin_reg.predict(news_test_data)
sgd_pred = sgd_reg.predict(news_test_data)
knn_pred = knn_reg.predict(news_test_data)
tree_pred = tree_reg.predict(news_test_data)


def printScores():
    global news_test_target, knn_pred, lin_pred, sgd_pred, tree_pred
    print("linear reg Mean squared error of a taught model: %.2f" %
        mean_squared_error(news_test_target, lin_pred))
    print("sgd reg Mean squared error of a taught model: %.2f" %
        mean_squared_error(news_test_target, sgd_pred))
    print("knn reg Mean squared error of a taught model: %.2f" %
        mean_squared_error(news_test_target, knn_pred))
    print("tree reg Mean squared error of a taught model: %.2f" %
        mean_squared_error(news_test_target, tree_pred))

    print('linear reg r2 score: %.2f' % r2_score(news_test_target,
lin_pred))
    print('sgd reg r2 score: %.2f' % r2_score(news_test_target,
sgd_pred))
    print('knn reg r2 score: %.2f' % r2_score(news_test_target,
knn_pred))
    print('tree reg r2 score: %.2f' % r2_score(news_test_target,
tree_pred))

printScores()


def plotPredictions():
  global news_test_target, knn_pred, lin_pred, sgd_pred, tree_pred

  fig, ax = plt.subplots(figsize=(30,20))
  size=40
```

```
    x = range(size)
    offset=1500
    ax.plot(x, lin_pred[offset:offset+size], label='linear', linestyle=(5, (10,
3)), linewidth='2')
    ax.plot(x, knn_pred[offset:offset+size], label='knn', linestyle=(0, (5, 1)),
linewidth='2')
        # ax.plot(x, sgd_pred[offset:offset+size], label='sgd')  # etc.
    ax.plot(x, tree_pred[offset:offset+size], label='tree', linestyle=(0, (3, 1,
1, 1)), linewidth='2')
    ax.plot(x, news_test_target.iloc[offset:offset+size], label='real', lin-
ewidth='3')
    ax.set_xlabel('sample id')
    ax.set_ylabel('sample value')
    ax.set_title("regressors comparision")
    ax.legend(prop={'size': 40})

plotPredictions()



pt = PolynomialFeatures(2, )
normalizer = Normalizer()
scaler = StandardScaler()

news_train_data, news_test_data, \
news_train_target, news_test_target = \
train_test_split(data_np,target_np, test_size=0.2, random_state=10)

news_train_poly = pt.fit_transform(news_train_data)
news_test_poly = pt.fit_transform(news_test_data)

normalized_news_train_poly = normaliz-
er.fit_transform(news_train_poly)
normalized_news_test_poly = normalizer.fit_transform(news_test_poly)

scaled_news_train_poly = scaler.fit_transform(news_train_poly)
scaled_news_test_poly = scaler.fit_transform(news_test_poly)
```

```
def fitAndScore(regressor,train, test, name):
    regressor.fit(train, news_train_target)
    prediction = regressor.predict(test)
    print("Mean squared error of a "+name+": %.2f" %
        mean_squared_error(news_test_target, prediction))
    score = regressor.score(test, news_test_target) #r2_score
    print("Variance score "+name+": %.2f" % score)


regressors = (lin_reg, knn_reg, tree_reg, sgd_reg)
names = ("lin_reg", "knn_reg", "tree_reg", "sgd_reg")

for reg, name in zip(regressors, names):
    print(name)
    fitAndScore(reg, news_train_poly, news_test_poly, name+" using pol-
ynomial features")
    fitAndScore(reg, normal-
ized_news_train_poly,normalized_news_test_poly, name+" using nor-
malized polynomial features")
    fitAndScore(reg, scaled_news_train_poly,scaled_news_test_poly,
name+" using scaled polynomial features")


lasso_r = Lasso(alpha=0.5, max_iter=5000,)
lasso_r.fit(news_train_poly, news_train_target)

print("Mean squared error of a linear moderl: %.2f" %
      mean_squared_error(news_test_target, las-
so_r.predict(news_test_poly)))
score = lasso_r.score(news_test_poly, news_test_target) #r2_score
print("Lasso regression variance score: %.2f" % score)


lasso_r = Lasso(alpha=0.5, max_iter=5000,)
lasso_r.fit(news_train_data, news_train_target)
```

```
pred = lasso_r.predict(news_test_data)

print("Mean squared error of a linear moderl: %.2f" %
      mean_squared_error(news_test_target, pred))
score = lasso_r.score(news_test_data, news_test_target) #r2_score
print("Lasso regression variance score: %.2f" % score)


for i in range(20):
    print("Prediceted: "+str(pred[i])+", real:"+str(news_test_target[i]))
```

Dodatek B

Kod programu - Klasyfikacja

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Logisticclfression
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import Normalizer
from sklearn.feature_selection import RFE
from sklearn.linear_model import Lasso

df = pd.read_csv('OnlineNewsPopularity.csv', header=0)
names = df.columns
stripped = [s.strip() for s in names]
df.columns=stripped


# Analiza eksploracyjna danych
```

```
# Odrzucanie url i timedelta jako nie prognozujące
df.drop(columns=['url', 'timedelta'],inplace=True)
df.describe()


# Podział na artykuły popularne i niepopularne - granicą będzie mediana.

shares_median = df['shares'].median()
df['popularity'] = df['shares'].apply(lambda x: 0 if x <shares_median else 1)


df.describe()


plt.style.use('default')
plt.hist(df['popularity'], bins=2, facecolor = '#2ab0ff', edgecol-
or='#0776a3', linewidth=0.5)
plt.xticks([0,1], ["unpopular", "popular"])


df.drop(columns=['shares'],inplace=True)


# Z analizy powyższych wykresów wynika, że większość z cech nie posiada
wielu danych odstających. Wyjątkiem jest cecha będąca naszym celem, czyli
shares, gdzie widać znaczy rozstrzał między wartościami.

data = df.iloc[:,:-1]
target = df.iloc[:,-1]
print(target.shape)
print(data.shape)


data.hist(bins=100, figsize=(100,100))
plt.show()
```

```
data_np = np.array(data, dtype=np.int16)
target_np = np.array(target, dtype=np.int16)


print("First news in database")
print(data_np[1,:])
print('---Mean---')
print(data_np.mean(axis=0))
print('--std---')
print(data_np.std(axis=0))


scaler = StandardScaler()

scaled_data = scaler.fit_transform(data_np)


print("First news in database")
print(scaled_data[1,:])
print('---Mean---')
print(scaled_data.mean(axis=0))
print('--std---')
print(scaled_data.std(axis=0))


news_train_data, news_test_data, \
news_train_target, news_test_target = \
train_test_split(scaled_data,target, test_size=0.2, random_state=10)


print("Training dataset:")
print("news_train_data:", news_train_data.shape)
print("news_train_target:", news_train_target.shape)

print("Testing dataset:")
print("news_test_data:", news_test_data.shape)
```

```
print("news_test_target:", news_test_target.shape)


def printCrossValScore(name, sc):
    print("Cross Validations scores for ", name)
    print("Wyniki: ", sc)
    print("Średnia: ", sc.mean())
    print("Odchylenie standardowe: ", sc.std())


log_clf = LogisticRegression()
scores = cross_val_score(log_clf, news_train_data, news_train_target,
scoring='accuracy', cv=5)
print(scores)


printCrossValScore("Linear Regression", scores)


sgd_clf = SGDClassifier()
scores = cross_val_score(sgd_clf, news_train_data, news_train_target,
scoring='accuracy', cv=5)
print(scores)


printCrossValScore("SGD Classification", scores)


tree_clf = DecisionTreeClassifier()
scores = cross_val_score(tree_clf, news_train_data, news_train_target,
scoring='accuracy', cv=5)
print(scores)


printCrossValScore("Decision Tree Classification", scores)
```

```
knn_clf = KNeighborsClassifier()
scores = cross_val_score(knn_clf, news_train_data, news_train_target,
scoring='accuracy', cv=5)
print(scores)


printCrossValScore("KNN Classification", scores)


param_grid = {
    # 'n_neighbors': [3,5,7,10,15], 'weights':['uniform','distance'],
'leaf_size':[15,30,60,90]
    'n_neighbors': [10,15], 'weights':['uniform','distance'], 'leaf_size':[5,15,30]
}

grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, scor-
ing="accuracy", cv=5, return_train_score=True, verbose=3)
grid_search.fit(news_train_data, news_train_target)


print(grid_search.best_score_)
print(grid_search.best_params_)
print(grid_search.best_estimator_)


log_clf.fit(news_train_data, news_train_target)
sgd_clf.fit(news_train_data, news_train_target)
tree_clf.fit(news_train_data, news_train_target)
knn_clf = grid_search.best_estimator_
knn_clf.fit(news_train_data, news_train_target)


log_pred = log_clf.predict(news_test_data)
sgd_pred = sgd_clf.predict(news_test_data)
knn_pred = knn_clf.predict(news_test_data)
tree_pred = tree_clf.predict(news_test_data)
```

```
def printScores():
    global news_test_target, knn_pred, log_pred, sgd_pred, tree_pred

    print('linear clf r2 score: %.2f' % accuracy_score(news_test_target,
log_pred))
    print('sgd clf r2 score: %.2f' % accuracy_score(news_test_target,
sgd_pred))
    print('knn clf r2 score: %.2f' % accuracy_score(news_test_target,
knn_pred))
    print('tree clf r2 score: %.2f' % accuracy_score(news_test_target,
tree_pred))

printScores()


def plotConfMatrix(name, y_pred):
  global news_test_target
  cm=confusion_matrix(news_test_target,y_pred)
  conf_matrix=pd.DataFrame(data=cm,columns=['Predicted: Unpopu-
lar','Predicted: Popular'],index=['Actual: Unpopular','Actual: Popular'])
  plt.figure(figsize = (8,5))
  sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
  plt.title(name)
  plt.show()


plotConfMatrix("logistic regression", log_pred)
plotConfMatrix("sgd classification", sgd_pred)
plotConfMatrix("knn classification", knn_pred)
plotConfMatrix("tree classification", tree_pred)


pt = PolynomialFeatures(2, )
normalizer = Normalizer()
scaler = StandardScaler()
```

```
news_train_poly = pt.fit_transform(news_train_data)
news_test_poly = pt.fit_transform(news_test_data)

normalized_news_train_poly = normaliz-
er.fit_transform(news_train_data)
normalized_news_test_poly = normalizer.fit_transform(news_test_data)

scaled_news_train_poly = scaler.fit_transform(news_train_data)
scaled_news_test_poly = scaler.fit_transform(news_test_data)


def fitAndScore(clf,train, test, name):
    clf.fit(train, news_train_target)
    score = clf.score(test, news_test_target) #r2_score
    print("Accuracy score "+name+": %.2f" % score)


classificators = (log_clf, knn_clf, tree_clf, sgd_clf)
names = ("log_clf", "knn_clf", "tree_clf", "sgd_clf")

for clf, name in zip(classificators, names):
    print(name)
    fitAndScore(clf, news_train_poly, news_test_poly, name+" using poly-
nomial features")
    fitAndScore(clf, normal-
ized_news_train_poly,normalized_news_test_poly, name+" using nor-
malized polynomial features")
    fitAndScore(clf, scaled_news_train_poly,scaled_news_test_poly,
name+" using scaled polynomial features")
```