Tworzenie dobrych zbiorów uczących

IO wykład 9, Tomasz Dzido

Wstęp

- Rozważymy najważniejsze techniki wstępnego przetwarzania danych, dzięki którym łatwiej nam będzie tworzyć dobre modele uczenia maszynowego:

  a) usuwanie/wstawianie brakujących wartości w zbiorze danych

  b) dopasowywanie kategoryzujących danych do algorytmów uczenia maszynowego

  c) dobór odpowiednich cech podczas konstruowania modelu.

Kwestia brakujących danych

- Przyczyną może być jakiś błąd w procesie gromadzenia informacji, niemożliwość wykorzystania pewnych form pomiarów albo pozostawienie pustych pól podczas uzupełniania formularza itp. Widzimy je jako puste pola lub *NaN* lub *Null*.
- Jeśli je zignorujemy, to większość narzędzi sobie nie poradzi lub zacznie generować nieprzewidywalne wyniki.
- Główne sposoby radzenia sobie z tym problemem to usuwanie wpisów z zestawu danych lub wstawianie danych i cech z innych przykładów.

Wykrywanie brakujących wartości

- Rozpatrzmy dokładnie konkretny przykład – program brakujacedane.py.
- Funkcja StringIO pozwala na wczytanie ciągu znaków związanych z tabelą csv_data do obiektu DataFrame tak, jakby ona była zwyczajnym plikiem csv zapisanym na dysku.
- Ciekawe funkcje: isnull().sum(), dropna(axis=0), dropna(axis=1), dropna(how='all’), dropna(thresh=4), dropna(subset=['C’]).
- Wady takiego postępowania: usunięcie zbyt dużej liczby próbek, usunięcie zbyt dużo ważnych kolumn (cech). Potrzebne są techniki interpolacji.

Wstawianie brakujących danych

- Jedną z najpopularniejszych metod interpolacji jest imputacja z użyciem średniej (ang. mean inputation) – zastępujemy brakującą wartość średnią wyliczoną na podstawie całej kolumny cech.

SimpleImputer(missing_values=np.nan, strategy='mean’) z klasy SimpleImputer.

- Zamiast mean może być median, most_frequent
- Analiza przykładu: brakujacedane.py

Wstawianie brakujących danych – c.d

- To bardzo ciekawy i obszerniejszy temat. Wystarczy spojrzeć na:

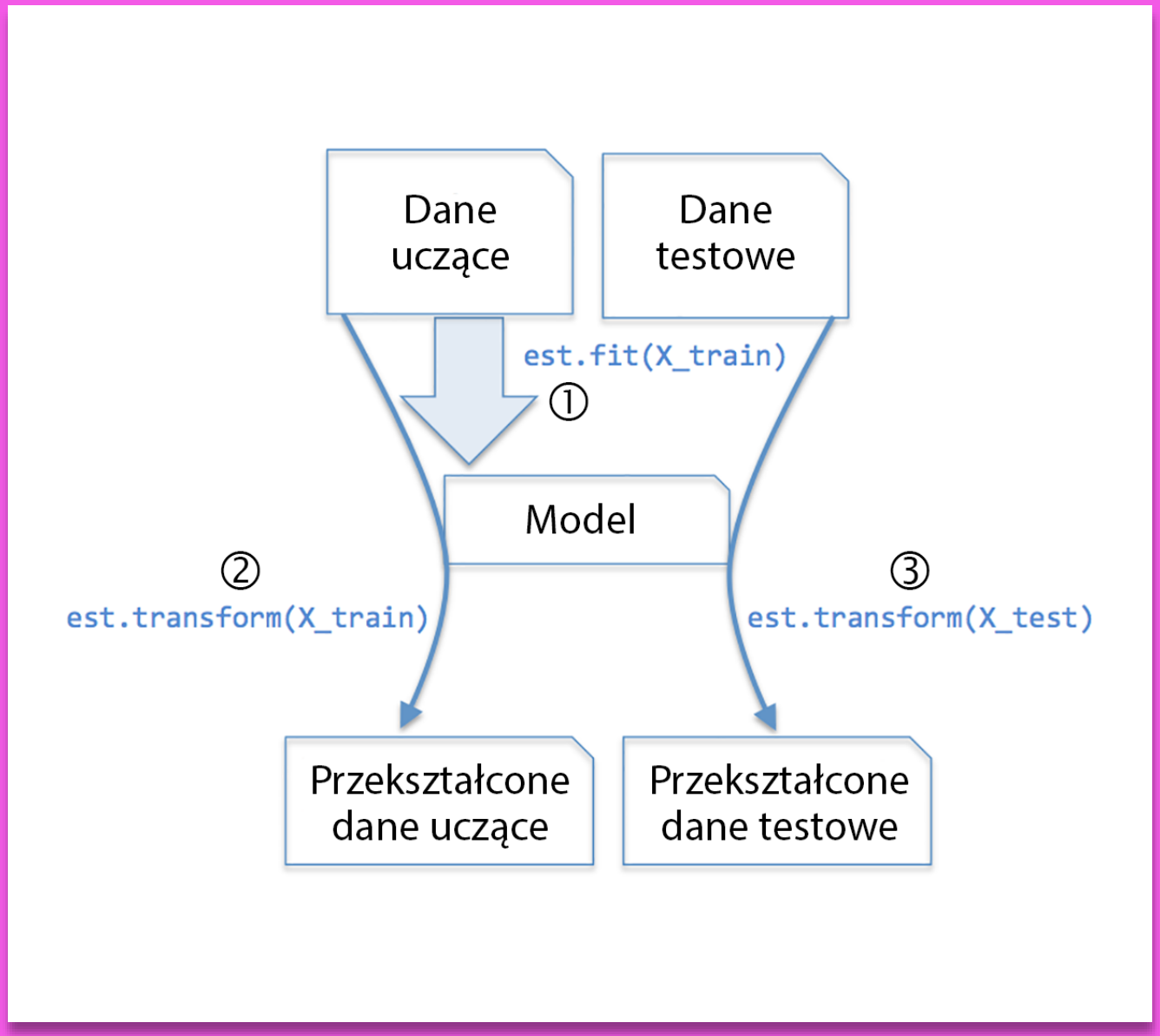
https://scikit-learn.org/stable/modules/impute.html#impute

- Rozpatrzmy taki przykład: (z powyższej strony)

```
>>> import numpy as np
>>> from sklearn.impute import KNNImputer
>>> nan = np.nan
>>> X = [[1, 2, nan], [3, 4, 3], [nan, 6, 5], [8, 8, 7]]
>>> imputer = KNNImputer(n_neighbors=2, weights="uniform")
>>> imputer.fit_transform(X)
array([[1. , 2. , 4. ],
       [3. , 4. , 3. ],
       [5.5, 6. , 5. ],
       [8. , 8. , 7. ]])
```

Klasy transformujące

- Należy do nich klasa SimpleImputer. Posiada 2 podstawowe metody: fit i transform.
- Metoda fit służy do dopasowywania parametrów poprzez dane uczące, a metoda transform wykorzystuje te parametry do modyfikowania danych.
- Przekształcana tabela danych musi zawierać taką samą liczbę cech jak tabela danych wykorzystana do uczenia algorytmu.
- Estymatory (takie jak poznane klasyfikatory) wykorzystują metodę predict ale można używać również metody transform.



Źródło: Raschka, Mirjalili: Python. Uczenie maszynowe, Helion 2019

Przetwarzanie danych kategoryzujących

- Wyróżniamy 2 rodzaje takich danych: cechy nominalne (np. kolor koszulki) i cechy porządkowe (np. rozmiar koszulki).

  Przykład: kategoryczne.py
- Dobrze jest dokonać mapowania cech porządkowych (przekształcenia wartości kategoryzujących w postać liczb całkowitych). Musimy to zrobić własnoręcznie. Proces można odwrócić (w obu sytuacjach metoda map).
- Algorytmy klasyfikacji nie korzystają z informacji porządkowych zawartych w etykietach klas, a wiele algorytmów wymaga by te etykiety były kodowane w postaci liczb całkowitych. Kodowania można dokonać ręcznie lub wykorzystać zaimplementowane mechanizmy (kategoryczne.py).

Kodowanie „gorącojedynkowe” cech nominalnych (z użyciem wektorów własnych)

- Kontynuujmy przykład. Co się stanie gdy cechę nominalną *kolor* zakodujemy tak jak *etykiety klas*? – dostaniemy: 0 (niebieski), 1 (zielony) i 2 (czerwony). Algorytm uczący będzie teraz zakładał, że zielony jest większy od niebieskiego….
- Rozwiązaniem tego problemu jest kodowanie „gorącojedynkowe” (ang. one-hot encoding). Wprowadzamy sztuczne cechy (ang. dummy feature) dla każdej unikatowej wartości w kolumnie cechy nominalnej. U nas wykorzystamy wartości binarne do wskazywania danego koloru próbki np. Niebieski=1, Zielony=0, Czerwony=0 dla próbki niebieskiej. Korzystamy z OneHotEncoder, ColumnTransformer. Patrz kategoryczne.py

Kodowanie „gorącojedynkowe” –c.d.

- Ciekawą opcją jest zastosowanie metody *get_dummies* z biblioteki *pandas* (przekształcone zostaną jedynie kolumny zawierające ciągi znaków).
- Ważne podczas takiego kodowania musimy pamiętać, że wprowadza ono współliniowość, cechy są ze sobą silnie skorelowane (odwrócenie macierzy staje się skomplikowane, mogą wystąpić niestabilne numerycznie oszacowania). Wystarczy usunąć jedną kolumnę – zauważmy, że nie tracimy żadnych informacji (usuńmy np. kolumnę *Kolor_Niebieski*).
- Przykład: kategoryczne.py

Skalowanie cech

- Większość algorytmów ML wymaga skalby działać skutecznie (wyjątkiem drzewa decyzyjne, lasy losowe).
- Gdy mamy 2 cechy: jedną mierzoną w skali 1:10, a drugą w skali 1:100000, to rozpatrując funkcję sumy kwadratów błędów czy odległości w algorytmie KNN, wartości drugiej cechy zdominują.
- 2 popularne metody: normalizacja i standaryzacja. Pojęcia są często luźno stosowane w różnych dziedzinach nauki.
- Normalizacja – najczęściej skalowanie cech do zakresu [0,1], co stanowi specjalny przypadek skalowania min.-max. Wzór:

$$x_{norm}^{(i)} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}}$$

Skalowanie cech – c.d.

- Standaryzacja znacznie praktyczniejsza w wielu algorytmach uczenia maszynowego, zwłaszcza optymalizacyjnych np. gradientu prostego. Algorytmy regresji logistycznej inicjują wagi o zerowej lub losowej, bardzo zbliżonej do 0 wartości. Chcielibyśmy zaś wprowadzić do kolumn średnią 0 przy odchyleniu standardowym 1, dzięki czemu dostajemy rozkład normalny, ułatwiający uczenie wag.
- Standaryzacja zachowuje informacje na temat odstających próbek, zmniejsza wrażliwość algorytmu na krańcowe przypadki, w przeciwieństwie do skalowania min.-max. (ograniczony zakres wartości). Standaryzacja wygląda następująco:

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x},$$

μ_x to średnia próbek a σ_x to odchylenie standardowe.

Skalowanie cech - przykład

- Możemy przeprowadzić ręcznie standaryzację i normalizację lub wykorzystać interfejs scikit-learn (klasy MinMaxScaler oraz StandardScaler). Wykorzystamy zestaw danych Wine, który stanowi otwarty zbiór informacji dostępny w repozytorium UCI, składa się ze 178 próbek win opisywanych 13 cechami.

Przykład: skalowanie.py

- Ważne: te klasy wprowadzamy tylko raz – wobec danych uczących – a następnie wykorzystujemy te wyniki do przekształcenia danych testowych lub każdej nowej próbki.

Dobór odpowiednich cech

- Przyczyną przetrenowania modelu (mówimy, że cechuje go duża wariancja) jest zbyt duża złożoność modelu wobec wykorzystywanych danych uczących. Problem ten możemy rozwiązać poprzez:

  a) zwiększenie ilości danych uczących (są różne techniki)

  b) wprowadzenie kar za złożoność poprzez regularyzację (było wcześniej)

  c) dobór prostszego modelu zawierającego mniej parametrów

  d) zmniejszenie wymiarowości danych.

- Na tym wykładzie zajmiemy się punktem d). Zauważmy z poprzedniego wykładu, że regularyzacja L1 może być traktowana jako jedna z technik wyboru cech. Istnieją jednak inne sposoby.

Algorytmy sekwencyjnego wyboru cech

- Wybór cech to wyznaczanie podzbioru pierwotnych cech i jest to jeden z podstawowych sposobów redukcji wielowymiarowości (inny to odkrywanie cech gdy pozyskujemy ze zbioru cech informacje pozwalające stworzyć nową podprzestrzeń cech).
- Na tym wykładzie klasyczne algorytmy doboru cech.
- Sztandarowym przykładem zachłannego, sekwencyjnego algorytmu wyboru cech jest algorytm sekwencyjnej selekcji wstecznej (ang. Sequential Backward Selection – SBS), w którym staramy się zredukować wymiarowość początkowej d-wymiarowej przestrzeni cech do podprzestrzeni k-wymiarowej (k<d) przy minimalnym rozpadzie skuteczności klasyfikatora. Chcemy poprawić skuteczność obliczeniową a może nawet wpłynąć na siłę predykcyjną przetrenowanego modelu.

Algorytm SBS – idea działania

- Algorytm sekwencyjnie usuwa cechy z zapełnionego podzbioru cech aż do wprowadzenia odpowiedniej ich liczby do nowego podzbioru.
- Konieczne jest zdefiniowanie funkcji kryterialnej J (które cechy usuwać) np. jako różnicę w skuteczności klasyfikatora przed pozbyciem się danej cechy i po jej usunięciu. Funkcja będzie maksymalizowała tę wartość kryterialną czyli na końcu każdego etapu eliminujemy cechę, której usunięcie w najmniejszym stopniu obniża skuteczność modelu.

Algorytm SBS - działanie

1. Inicjacja algorytmu przy zadanym parametrze $k = d$, gdzie d oznacza wymiarowość pełnej przestrzeni cech $\boldsymbol{x}_d$.
2. Określenie cechy x^- maksymalizującej funkcję kryterialną $\boldsymbol{x}^- = \arg\max J\,(\boldsymbol{X}_k - \boldsymbol{x})$, gdzie $\boldsymbol{x} \in \boldsymbol{X}_k$.
3. Usunięcie cechy x^- ze zbioru cech: $\boldsymbol{x}_{k-1} = \boldsymbol{X}_k - \boldsymbol{x}^-$; $k = k-1$.
4. Zakończenie działania, jeżeli k jest równe liczbie wymaganych cech, w przeciwnym wypadku powrót do etapu 2.

Źródło: Raschka, Mirjalili: Python. Uczenie maszynowe, Helion 2019

Co interesujące, algorytm SBS nie został jeszcze zaimplementowany w scikit-learn. Program AlgorytmSBS.py zawiera jego implementację wraz z komentarzami.

Algorytm SBS – sprawdzenie działania

- Sprawdzamy implementację algorytmu SBS wykorzystującej klasyfikator KNN:

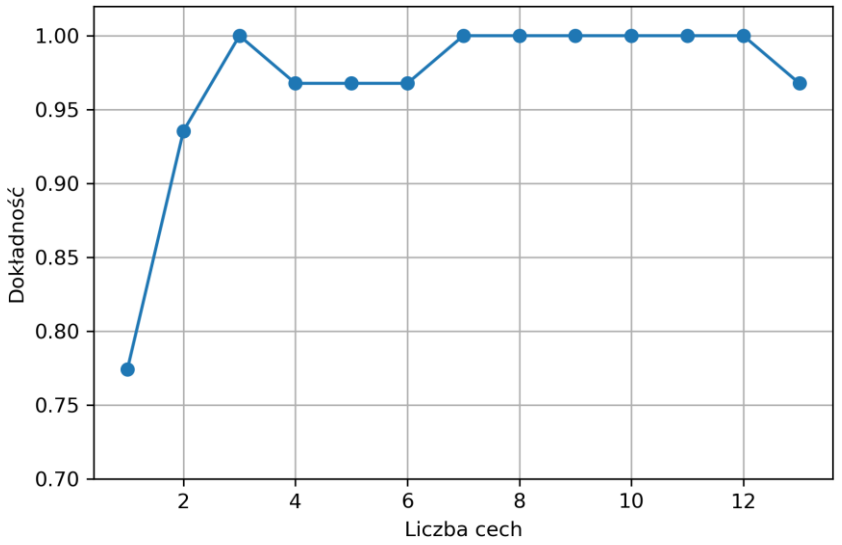
```
knn = KNeighborsClassifier(n_neighbors=5)
sbs = SBS(knn, k_features=1)
sbs.fit(X_train_std, y_train)
```

- Metoda fit wygeneruje ponownie podział na dane do uczenia i walidacji, przez to sprawiamy, że pierwotny zestaw testowy nie zostaje dołączony do danych uczących.
- Algorytm na każdym etapie zapisuje punktację najlepszego podzbioru cech – warto wygenerować wykres klasyfikatora KNN obliczanego na zestawie walidacyjnym.

Algorytm SBS – przykład c.d.

- Dokładność klasyfikatora KNN poprawiła się wraz ze zmniejszaniem liczby cech, co prawdopodobnie ma związek z redukowaniem klątwy wielowymiarowości.
- 100% dokładności dla

  k = [3,7,8,9,10,11,12]



Źródło: Raschka, Mirjalili: Python. Uczenie maszynowe, Helion 2019

Algorytm SBD – dalsza analiza przykładu

- Warto sprawdzić jak wygląda najmniejszy podzbiór cech (k=3) gwarantujący taką dobrą skuteczność podczas analizowania danych walidacyjnych (interesuje nas dziesiąta pozycja atrybutu sbs.subsets_)
- Na koniec warto sprawdzić skuteczność KNN wobec pierwotnego zestawu testowego i wybranego trójelementowego podzbioru cech.
- Jaki wniosek? Nie zdołaliśmy zwiększyć wydajności modelu KNN (te 3 cechy nie dostarczają tak wielu informacji jak pierwotny zestaw danych), udało nam się jednak zmniejszyć rozmiar zestawu danych, co może się przydać gdzie danych jest dużo (tutaj mało danych, podatne na losowość etc.). Dzięki zredukowaniu liczby cech uzyskaliśmy prostszy model.

Inne algorytmy wyboru cech

- Można wymienić rekurencyjną eliminację wsteczną bazującą na wagach cech, metody drzew polegające na doborze cech pod kątem ich istotności, czy jednoczynnikowe testy statystyczne.
- Podsumowanie i graficzne przykłady: (ciekawa implementacja oparta na L1 i wiele innych)

https://scikit-learn.org/stable/modules/feature_selection.html