Dydaktyka Informatyki 14(2019)
ISSN 2083-3156; e-ISSN 2543-9847
DOI: 10.15584/di.2019.14.13
http://www.di.univ.rzeszow.pl

Uniwersytet Rzeszowski

Jacek WOŁOSZYN

ORCID: 0000-0003-4340-9853, Dr inż., Uniwersytet Technologiczno-Humanistyczny w Radomiu, Wydział Informatyki i Matematyki, Katedra Informatyki, ul. Malczewskiego 29, 26-600 Radom, e-mail: jacek@delta.pl

WYKORZYSTANIE TECHNIKI ADABOOST W MODELACH OPARTYCH NA REGRESJI Z WYKORZYSTANIEM SZTUCZNEJ INTELIGENCJI

USE OF ADABOOST TECHNIQUE IN REGRESSION BASED MODELS USING ARTIFICIAL INTELLIGENCE

Słowa kluczowe: AdaBoost, sztuczna inteligencja.
Keywords: AdaBoost, artificial intelligence.

Streszczenie

Artykuł opisuje wykorzystanie sztucznej inteligencji w obliczeniach regresyjnych. Modele zbudowane w sposób tradycyjny oparte na klasycznych założeniach dostarczają możliwie precyzyjnych informacji. Sam proces budowy modelu opiera się na wstępnym wyborze zmiennych wykorzystanych do jego tworzenia. Umiejętna selekcja zmiennych ma istotny wpływ na uzyskane parametry. Wykorzystując rozwiązanie przedstawione w tej pracy otrzymujemy model ze wstępnie dobranym optymalnym zbiorem treningowym.

W kolejnych rozdziałach omówiono istotę analityki predykcyjnej, proces uczenia maszynowego, budowę drzewa decyzyjnego, pokazano przykład regresji wykorzystującego AdaBoost.

Abstract

This article describes the use of artificial intelligence in regression calculations. Models built in a traditional manner based on classical assumptions provide the most precise information possible. The model building process itself is based on the initial selection of variables used to create it. Skillful selection of variables has a significant impact on the obtained parameters. Using the solution presented in this work, we get a model with a pre-selected optimal training set.

The following chapters discuss the essence of predictive analytics, the machine learning process, the construction of a decision tree, an example of regression using AdaBoost.

Wstęp

Budowanie modelu ekonometrycznego tradycyjnym sposobem opierało się w większości przypadków na wykorzystaniu techniki regresji. Otrzymany model po spełnieniu wcześniejszych warunków związanych z doborem danych do modelu, współliniowością, heteroscedastycznością i innych można wykorzystać do celów predykcji. W przypadku utworzenia kilku modeli należało wybrać najlepszy korzystając z kryteriów wyboru modelu Mallowsa, Akaike oraz innych.

Wraz z rozwojem techniki obliczeniowej, wykorzystując wysoką wydajność zasobów systemowych maszyn oraz zaawansowane algorytmy obliczeniowe, model tradycyjny można zastąpić modelem maszynowym z wykorzystaniem algorytmów sztucznej inteligencji. Rozwiązanie to pozwala na szybkie i precyzyjne utworzenie optymalnego modelu, testowaniu jego działania na dołączonym zestawie danych, a co najważniejsze – dynamicznej zmianie parametrów w funkcji napływu nowych danych.

Do implementacji wykorzystano język Python[1] w wersji 3, bibliotekę NumPy[2] wykorzystywaną jako podstawa w uczeniu maszynowym. Za jej pomocą można przeprowadzać efektywne operacje na strukturach danych takich jak tensor, wektor, macierz. Same procedury uczenia maszynowego[3] zostały wykorzystane z biblioteki scikit-learn[4]. Udostępnia ona wiele algorytmów z dziedziny uczenia nadzorowanego i nienadzorowanego.

Analityka predykcyjna

Istotą analityki predykcyjnej jest tworzenie modeli na podstawie zgromadzonych historycznych informacji. W tradycyjnej metodzie tworzenia modeli proces ten był żmudny, długotrwały. Drobna zmiana w danych, z których był utworzony model, nawet o kilka dodatkowych rekordów może pociągnąć za sobą modyfikację właściwości współczynników modelu. Szybko napływające dane wymagają korekty modelu, a modyfikacja modelu była niezmiernie trudna, a nawet niemożliwa do wykonania w krótkim czasie. Wraz z rozwojem możliwości obliczeniowych systemów komputerowych i wykorzystaniu algorytmów sztucznej inteligencji pojawiły się niespotykane do tej pory możliwości. Utwo-

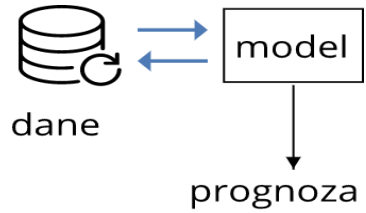
[1] M. Goodrich, R. Tamassia, M. Goldwasser, *Data Structures and Algorithms in Python*, Wiley 2013.

[2] Y. Hilpisch, *Derivatives Analytics with Python*, Wiley 2015.

[3] S. Raschka, V. Mirjalili, *Python Machine Learning Packt 2017.*

[4] F. Pedregosa, V. Gaël, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, É. Duchesnay; *12 Scikit-learn: Machine Learning in Python* (Oct):2825–2830, 2011.

rzone modele na podstawie zgromadzonych danych potrafią szybko i wydajnie reagować na przychodzące informacje, a tym samym bardzo precyzyjnie prognozować przyszłe zjawiska z rys. 1.



Rys. 1. Proces uczenia maszynowego

Przykładem działania uczenia maszynowego[5] może być system pokazywania reklam na stronach www, z których codziennie korzystamy. Na stronach pokazują się reklamy. Z jednej strony my jako użytkownicy jesteśmy klientami, dla których te reklamy się pokazują. Stanowimy dla reklamodawców pewną wartość, każdy inną w zależności od zainteresowań, częstotliwości zakupów i innych. Z drugiej strony reklamodawcy są skłonni zapłacić za możliwość wyświetlenia reklamy. A ponieważ reklamodawców jest sporo, to wygrywa ten, który jest skłonny zapłacić najwięcej. W związku z tym algorytm musi podjąć decyzję w bardzo krótkim czasie, porównywalnym do czasu uruchamiania strony o tym komu i jaką reklamę pokazać.

Budowa modelu opartego na DecisionTree Regressor

Drzewo decyzyjne jest strukturą, która dzieli zbiór danych na podzbiory i dokonuje prostych wyborów na każdym poziomie. Idąc tą drogą można uzyskać końcowe rozwiązania, które charakteryzują się optymalnym wyborem. Drzewa decyzyjne są tworzone z wykorzystaniem algorytmów szkoleniowych. Sam proces decyzyjny rozpoczyna się w węźle głównym na szczycie drzewa, a każdy węzeł jest regułą decyzyjną. Algorytmy podejmują decyzje na podstawie relacji między danymi wejściowymi a etykietami docelowymi na danych treningowych. Wartości danych wejściowych są wykorzystywane do oszacowania wartości wyjściowej.

Proces konstruowania drzew decyzyjnych

Drzewa decyzyjne to modele uczenia maszynowego w formie struktury drzewa. Są szybkie w działaniu i nie wykorzystują nadmiernie zasobów systemowych, dlatego są chętnie wykorzystywane. Każdy węzeł niebędący liściem

[5] A. Boschetti, L. Massaron, *Python Podstawy nauki o danych*, Helion, Gliwice 2017.

jest decydentem. Decyzja to wykonanie kolejnego kroku na podstawie przeprowadzonego testu. Budowa drzewa decyzyjnego natomiast polega na zmianie wartości entropii na mniejszą. Przechodząc w dół drzewa zmniejszamy entropię od całkowitej niepewności w kierunku pewności.

Jeśli zdefiniujemy entropię jako:

$$\text{entropia } (X) = -\sum p_i \text{ i } \log 2\ p_i$$

gdzie p_i odnosi się do prawdopodobieństwa występowania w zbiorze danych

Tak więc można powiedzieć, że w zbiorze 60 elementów, składających się z trzech typów elementów każdy element pojawia się 20 razy. Można w tym przypadku powiedzieć, że entropia jest wysoka, ponieważ istnieje duża niepewność. Jeśli natomiast element typu pierwszego pojawi się w zbiorze 58 razy, wówczas niepewność jest niska. Jeśli losowo zostanie wybrana w tym przypadku próbka z dużym prawdopodobieństwem można przewidzieć, że będzie to element typu pierwszego.

Jak w takim razie można zmniejszyć entropię?

Rozważmy przykład zbioru danych z 60 elementami, w których typy elementów pojawiają się odpowiednio 14, 27, 19 razy.

Całkowita entropia:

$$\text{entropia} = -(14/60) * \log(14/60) - (27/60) * \log(27/60) - (19/60) \log(19/60) = 1{,}537$$

Dzielimy ten zestaw danych na dwie części aby zmniejszyć całkowitą entropię.

lewe poddrzewo = [5, 21, 11]
prawe poddrzewo = [9, 6, 8]

Liczymy entropię dla poszczególnych gałęzi

$$\text{lewa entropia} = -(5/37) \log(5/37) - (21/37) \log(21/37) - (11/37) \log(11/37) = 1{,}374$$
$$\text{prawa entropia} = -(9/23) \log(9/23) - (6/23) \log(6/23) - (8/23) \log(8/23) = 1{,}565$$

Całkowita entropia dla dwóch gałęzi, liczymy poprzez ważone sumowanie.

$$\text{całkowita entropia} = (37/60) * 1{,}374 + (23/60) * 1{,}565 = 1{,}447$$

Różnica pomiędzy początkową entropią, a końcową wynosi zatem:

$$\text{zysk informacyjny} = 1{,}537 - 1{,}447 = 0{,}09$$

Oznacza to, że jeśli podzielimy zestaw danych w taki właśnie sposób zwiększymy zysk informacyjny o 0.09

Z biblioteki sklern można wykorzystać funkcję DecisionTreeClassifier(**params), aby utworzyć klasyfikator drzewa decyzyjnego. Aby ocenić wydajność należy skorzystać z sklern.classification_report. Dla regresji odpowiednio DecisionTreeRegressor.

Przykład wykorzystania AdaBoost

AdaBoost to algorym pozwalający z dużej liczby klasyfikatorów uzyskać jeden o optymalnych parametrach.

Poniższy przykład pokaże wszystkie te zasady zastosowane do oszacowania cen domów. W celu rozwiązania tego problemu użyjemy regresora drzewa decyzyjnego z AdaBoost. W drzewie decyzyjnym, jak wcześniej wspomniano, każdy węzeł podejmuje prostą decyzję, która przyczynia się do ostatecznego wyniku. Węzły liścia reprezentują wartości wyjściowe, a gałęzie przedstawiają decyzje pośrednie, które zostały podjęte na podstawie cech wejściowych AdaBoost oznacza adaptacyjne wspomaganie.

Jest to technika, która służy do zwiększenia dokładności wyników z innego systemu dołączy wyjścia z różnych wersji algorytmów zwanych swoimi uczniami używając ważonego sumowania, aby uzyskać końcowy wynik. Informacje zebrane na każdym etapie algorytmu AdaBoost są ponownie wprowadzane do systemu, aby uczący się na późniejszych etapach koncentrował się na próbkach szkoleniowych, które są trudne do sklasyfikowania. W ten sposób zwiększa się dokładność systemu. Korzystając z AdaBoost dopasowujemy regressor do zestawu danych obliczamy błąd, a następnie ponownie dopasowujemy go do tego samego zestawu danych na podstawie oszacowanego błędu. Możemy myśleć o tym jako precyzyjnym do strojeniu modelu do osiągnięcia pożądanej dokładności.

Otrzymujemy zbiór danych, który zawiera różne parametry, jakie wpływają na cenę domu.

Naszym celem jest oszacowanie relacji między tymi parametrami a ceną domu, abyśmy mogli użyć tego do oszacowania ceny przy nieznanych parametrach wyjściowych.

Do rozwiązania problemu zostanie wykorzystana biblioteka scikit-learn. Jest to biblioteka oferująca proste i wydajne narzędzia do eksploracji i analizy danych oferowany na licencji BSD. Uczenie maszynowe, które wykorzystuje zasoby systemów komputerowych wyszuka optymalne rozwiązanie bez wskazania konkretnych instrukcji, opierając się na wzorcach i wnioskowaniu.

W pierwszej kolejności należy zaimportować niezbędne biblioteki:

```
- numpy
- sklern - DecisionTreeRegressor,        AdaBoostRegressor, datasets
- sklern,metrics mean_squared_error,     explained_variance_score
- sklern.utils shuffle
```

To warunki niezbędne do wykonania obliczeń. Dane można pobrać z bazy danych https://arcive.ics.uci.edu/ml/machine-learning-databases, która udostępnia bibliotekę z wieloma przykładami do wykorzystania. Każda pozycja obejmuje część opisową oraz część zawierającą właściwą bazę z danymi. W naszym przypadku część opisowa przedstawia się następująco.

1. CRIM – per capita crime rate by town
2. ZN – proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS – proportion of non-retail business acres per town
4. CHAS – Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX – nitric oxides concentration (parts per 10 million)
6. RM – average number of rooms per dwelling
7. AGE – proportion of owner-occupied units built prior to 1940
8. DIS – weighted distances to five Boston employment centres
9. RAD – index of accessibility to radial highways
10. TAX – full-value property-tax rate per $10,000
11. PTRATIO pupil-teacher ratio by town
12. B – 1000 (Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT – % lower status of the population
14. MEDV – Median value of owner-occupied homes in $1000's

Aby zaimportować dane należy użyć funkcji read_csv(param) lub skorzystać z funkcji udostępnianej przez bibliotekę scikit-learn. Datasets pozwala na bezpośrednie pobranie zbioru danych z repozytorium.

```
data = datasets.load_boston()
```

Ten zbiór danych należy podzielić na zmienne objaśniane i objaśniające. W tym przypadku zmienną objaśnianą jest MEDV – mediana wartości domów. Będziemy próbowali ustalić, od jakich parametrów i w jakim stopniu zależy cena domów. Aby przeprowadzany trening był niezależny od kolejności wprowadzanych rekordów zastosujemy funkcję shuffle.

```
X, y = shuffle(housing_data.data, housing_data.MEDV, random_state=7).
```

Tak przygotowane dane, dzielimy na część szkoleniową i część testową – odpowiednio 80% i 20%.

```
num_training = int(0.8 * len(X))
X_train, y_train = X[:num_training], y[:num_training]
X_test, y_test = X[num_training:], y[num_training:]
```

Celem szkolenia maszynowego jest utworzenie modelu, który rozpoznaje wcześniej niewidoczne zależności pomiędzy zmiennymi i przedstawia je w postaci uogólnionej.

Podział na dwie części wynika ze sposobu przeprowadzania procedury. Część szkoleniowa służy do szkolenia modelu, a część testowa do weryfikacji uzyskanego modelu, jego generalizacji.

```
dt_regressor = DecisionTreeRegressor(max_depth=4)
dt_regressor.fit(X_train, y_train)
```

Tworzymy i szkolimy regresor. Do utworzenia regresora została użyta funkcja DecisionTreeRegressor omówiona w poprzednim rozdziale.

Dla porównania utworzymy także zmienną ab_regressor, która wykorzystuje dodatkowo AdaBoost.

```
ab_regressor = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
n_estimators=400, random_state=7)
ab_regressor.fit(X_train, y_train)
```

Mając tak przygotowane regresory można dokonać oceny ich wydajności. Aby to wykonać należy użyć funkcji predict(), która służy do przewidywania odpowiedzi na podstawie danych testowych.

Ocena wydajności regresora dt_regressor

```
y_pred_dt = dt_regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred_dt)
evs = explained_variance_score(y_test, y_pred_dt)
# Decision Tree performance
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))
```

Ocena wydajności regresora ab_regressor()

```
y_pred_ab = ab_regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred_ab)
evs = explained_variance_score(y_test, y_pred_ab)
# AdaBoost performance
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))
```

Otrzymane wyniki:

Decision Tree performance

Mean squared error = 14.79
Explained variance score = 0.82

AdaBoost performance

Mean squared error = 7.66
Explained variance score = 0.91

Analiza wyników potwierdza, że użycie funkcji AdaBoost zdecydowanie zwiększa precyzję działania modelu. Błąd ma niższą wartość i wynosi 7.66, a wariancja jest bliższa jedności.

Podsumowanie

Celem modelu analitycznego jest identyfikacja związków występujących między rozpatrywanymi zjawiskami w przeszłości. Precyzyjnie utworzony model pozwala dokładnie wyjaśnić mechanizmy działające na objaśnianą zmienną, a tym samym poznać wektory operujące największą siłą na badane zjawisko.

Regresor AdaBoost jest meta-estymatorem, zaczyna działanie na pełnym zestawie danych. W każdym iteracyjnym przejściu mierzony jest błąd, im większy błąd, tym mniejsza waga klasyfikatora. Pozwala to na uzyskanie optymalnego zestawu danych do treningu i tym samym znaczne zwiększenie wydajności regresora drzewa decyzyjnego, a co za tym idzie – trafność decyzji.

Dzięki takiemu podejściu otrzymujemy model bardzo dobrej jakości, za pomocą którego możemy generować dokładne prognozy. Wykorzystanie wyników prognozy umożliwia podjęcie optymalnej decyzji w wielu dziedzinach przemysłu jak i w życiu codziennym.

Bibliografia

Boschetti A., Massaron L., *Python Podstawy nauki o danych*, Helion, Gliwice 2017.

Hilpisch Y., *Derivatives Analytics with Python*, Wiley 2015.

Pedregosa F., Varoquaux G., Gramfort A., Michel V., Goodrich M., Tamassia R., Goldwasser M., *Data Structures and Algorithms in Python*, Wiley 2013.

Raschka S., Mirjalili V., *Python Machine Learning Packt 2017*.

Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M., Duchesnay É., *12 Scikit-learn: Machine Learning in Python* (Oct):2825−2830, 2011.