AutoML: Projekt 2 - System AutoML

Aleksandra Idczak, Krzysztof Maślak, Jakub Winiarski

1 Wstęp

Celem naszego projektu było stworzenie zautomatyzowanego systemu uczenia maszynowego, który łączy wysoką skuteczność predykcyjną z różnorodnością stosowanych algorytmów. Poniżej przedstawiamy nasze działania i wybory, obejmujące selekcję modeli bazowych, procedury preprocessingu oraz strategie tworzenia ensemblingów. W raporcie zawarliśmy także empiryczną weryfikację skuteczności systemu i porównanie z uznanymi pakietami, takimi jak MLjar oraz H2O AutoML. W celu uzyskania powtarzalności wyników wszędzie ustawialiśmy ziarna generatorów.

2 Wybór modeli bazowych

Wzorując się na systemie AutoGluon, staraliśmy się połączyć strategię wyboru najlepszych modeli pod kątem wyników, ze strategią wyboru różnorodnych modeli. Badania pokazują, że zestaw modeli działających w odmienny sposób daje zazwyczaj stabilniejsze i lepsze wyniki niż próba znalezienia jednego idealnego ustawienia hiperparametrów. Tym bardziej, że w kolejnym kroku połączyliśmy modele bazowe w ensemblingi.

Procedura wybierania najlepszych modeli z "MementoML" wyglądała następująco. Najpierw wybieraliśmy algorytm - przykładowo "ranger", który oznaczał Random Forest. Następnie sprawdzaliśmy, dla których zbiorów danych autorzy "MementoML" przetestowali dużą ilość hiperparametrów. W kolejnym kroku dla modelu wybrano pierwszy sensowny zbiór danych i zagregowano wyniki względem średniego AUC (dla każdej pary zbiór danych, zestaw hiperparametrów, mamy 20 AUC wynikające z 20 podziałów na zbiór treningowy oraz testowy; pod słowem agregacja rozumiemy średnią AUC po wszystkich 20 podziałach). Na rysunku 1 przedstawiono przykładowy wynik takiej agregacji.

	param_index	mean_auc	n.trees	interaction.depth	n.minobsinnode	shrinkage	bag.fraction
646	1647	0.905858	4449	4	2	0.005023	0.894624
600	1601	0.905634	1147	5	6	0.015169	0.927830
682	1683	0.905634	5385	5	4	0.001868	0.832182
244	1245	0.905597	302	5	2	0.042994	0.765124
617	1618	0.905560	518	4	6	0.032494	0.744368
460	1461	0.905518	2432	4	4	0.006521	0.463220

Rysunek 1: Zagregowane wyniki dla Random Forest

Na tej podstawie wybrano pierwszą najlepszą konfigurację danego algorytmu na danym zbiorze danych. Następnie wybrano drugi zbiór danych, dla którego przetestowano zróżnicowane zestawy hiperparametrów i postępując tak samo wybrano kolejną konfigurację itd. Naszym kryterium była także różnorodność, dlatego też, przy wyborze najlepszych konfiguracji staraliśmy się nie duplikować podobnych zestawów hiperparametrów. Ponadto odrzuciliśmy modele bardzo kosztowne obliczeniowo - przykładowo odrzuciliśmy te Random Foresty, które mają więcej niż 600 drzew.

Nie wszystkie konfiguracje zostały dobrane z "MementoML". W przypadku algorytmu KNN trudno nam było wyciągnąć z tegoż zbioru sensowne konfiguracje, w związku z czym zastosowaliśmy kryterium różnorodności biorąc 4 różne ilości sąsiadów. Oczywiście nie liczyliśmy na to, że KNN okaże się najlepszym modelem, ale że w ensemblingach przyda się model o odmiennym typie, patrzący na dane z innej strony.

Zdecydowaliśmy się, że nasz system będzie się składał z 44 modeli bazowych. Podział tych modeli, przy którym wzięliśmy pod uwagę skuteczność modeli boostingowych, jest przedstawiony poniżej:

Na końcu raportu znajdują się tabelki przedstawiające wszystkie wybrane przez nas modele wraz ze wszystkimi wybranymi hiperparametrami.

Tabela 1: Rozkład 44 modeli bazowych

Typ Modelu	**Liczba konfiguracji**
GradientBoostingClassifier	13
CatBoost	11
XGBoost	6
RandomForestClassifier	6
ElasticNet	6
KNeighborsClassifier	4

3 Strategia wyboru najlepszego modelu

Strategia wyboru najlepszego modelu przedstawia się następująco. Najpierw liczymy średnie balanced accuracy (istotne było dla nas strojenie systemu pod wymaganą w projekcie metrykę) z 5-krotnej kroswalidacji dla każdego z 44 modeli bazowych. Następnie wybieramy najlepszy model dla każdego z typów algorytmu - jest ich 6. Jest to nasz punkt startowy do tworzenia trzech ensemblingów.

Pierwszy ensembling to Stacking Classifier. Jego modele podstawowe to najlepszy GBM, CatBoost, RandomForest, ElasticNet oraz KNN. Natomiast finalnym estymatorem jest XGBoost. Przy liczeniu predykcji z modeli podstawowych został zastosowany domyślny podział zbioru na 5 foldów, przy czym w odróżnieniu od domyślnych ustawień w sklearn zadbaliśmy o przemieszanie zbioru danych przed podziałem.

Monitorując wyniki naszego systemu na różnych zbiorach danych i jego wynikach na zbiorze treningowym i testowym, doszliśmy do wniosku, że XGBoost jako finalny estymator w Stacking Classifierze może być podatny na overfitting. Stąd pomysł na dodanie drugiego Stacking Classifiera.

Modele podstawowe drugiego Stacking Classifiera to najlepszy GBM, CatBoost, XGBoost, RandomForest, ElasticNet oraz KNN. Finalnym estymatorem jest ElasticNet. Tutaj tak samo jak wcześniej zadbaliśmy o losowość w podziale na 5 foldów przy predykcjach modeli podstawowych.

Trzeci ensembling to Voting Classifier z miękkim głosowaniem. Czynny udział biorą tutaj: najlepszy GBM, CatBoost, RandomForest oraz XGBoost.

Dla każdego z ensemblingów wykonujemy tak samo jak wcześniej 5-krotną kroswalidację, liczymy średnie balanced accuracy i porównujemy z wcześniejszymi wynikami modeli bazowych. Na koniec wybieramy najlepszy model pod kątem rozważanej metryki i trenujemy go na całych danych treningowych.

Niestety, ze względu na metodę *predict_proba*, musieliśmy zrezygnować z Voting Classifiera z twardym głosowaniem.

4 Preprocessing

Do preprocessingu danych zaimplementowaliśmy dwie funkcje - jedna zajmuje się zbiorem treningowym, a druga testowym.

Na zbiorze treningowym: usuwamy kolumny i wiersze, które zawierają wyłącznie braki danych, wiersze, gdzie brak danych występuje w zbiorze y i wiersze, gdzie ponad 70% kolumn ma braki danych. Braki danych w kolumnach kategorycznych uzupełniamy modą, a następnie przeprowadzamy One Hot Encoding. Dla kolumn numerycznych braki uzupełniamy medianą i wszystkie skalujemy.

Dla zbioru testowego upewniamy się, że dane zawierają te same kolumny (oraz, że są one w tej samej kolejności) co na zbiorze treningowym. Jeśli, którejś z kolumn brakuje to zostaje ona dodana i uzupełniona modą (dla zmiennej kategorycznej) lub medianą (dla zmiennej numerycznej). Następnie wykonujemy uzupełnienie braków danych i One Hot Encoding / skalowanie, z użyciem statystyk ze zbioru trenignowego.

5 Porównanie z innymi pakietami

Aby sprawdzić, czy nasz system AutoML daje dobre i powtarzalne wyniki przetestowaliśmy go na dziewięciu zbiorach danych oraz porównaliśmy z dwoma innymi pakietami - MLjar i H2O AutoML. Wszystkie dane poza zbiorami example (dane od prowadzących) i local pochodzą z OpenML. Podzieliliśmy je na zbiory treningowy i testowy w proporcji 7:3. Dla MLjar ustawiliśmy hiperparametry time_limit=180 i mode="Perform", a dla H2O AutoML time_limit=180. Tabela 2 przedstawia wyniki eksperymentu - wartości balanced accuracy na zbiorze testowym. Dodatkowo kolumna

avg ba train MiniAutoML pokazuje średnią wartość balanced accuracy dla najlepszego modelu podczas kroswalidacji na zbiorze treningowym.

Dane	avg ba train MiniAutoML	ba test MiniAutoML	ba test MLjar	ba test H2O AutoML
blood transfusion	0.6516	0.5523	**0.6295**	0.5886
credit g	0.6570	**0.7075**	0.6875	0.5700
example	0.5708	**0.5561**	0.5515	0.5037
kc2 classification	0.7610	0.7478	**0.7742**	0.6740
kr vs kp	0.9955	0.9957	**0.9959**	0.9935
local	0.7242	**0.7303**	0.7194	0.7019
ozone level 8hr	0.6629	0.6381	**0.7110**	0.6486
qsar biodeg	0.8493	0.8411	**0.8803**	0.8606
wdbc	0.9700	**0.9926**	0.9829	0.9756

Tabela 2: Tabela porównująca wartość balanced accuracy na zbiorze testowym dla naszego systemu AutoML i pakietów MLjar i H2O AutoML. Wartości są zaokrąglone do czwartego miejsca po przecinku.

Porównując wartości w drugiej i trzeciej kolumnie widzimy, że wartości metryki podczas treningu i na zbiorze testowym są podobne - nie zachodzi przeuczenie. Jedynie w przypadku zbioru blood transfusion nastąpił wyraźny spadek balanced accuracy. Najlepszym modelem wybranym przez MiniAutoML był w tym przypadku stacking łączący najlepszy model z każdej rodziny i meta learner XGBoost. Możliwe, że końcowy XGBoost za bardzo dopasował się do danych. To skłoniło nas do dodania do puli możliwych modeli stackingu z "prostszym" meta learnerem - regresją logistyczną.

Porównując wyniki trzech systemów, widzimy, że nasz nie odstaje jakością od pozostałych. Pięciokrotnie najlepszy okazał się MLjar, cztery razy nasz system, a H2O AutoML ani razu nie osiągnął najwyższego wyniku. Najważniejsze było dla nas sprawdzenie, że nasze rozwiązanie radzi sobie na różnych zbiorach danych, a wyniki są stabilne oraz porównywalne do oficjalnych pakietów. Zestawiając wyniki naszego systemu z pakietami MLjar i H2O AutoML warto pamiętać, że zapewne oba pakiety mogłyby osiągnąć lepsze wyniki, gdyby czas treningu był dłuższy i hiperparametry (np. "tryb" działania AutoML) byłyby lepiej wyspecyfikowane. Natomiast ważne, że przy podobnym czasie trenowania jaki zużywa nasze rozwiązanie i prostych ustawieniach początkowych pakietów jakość modeli jest zbliżona.

6 Wnioski

Zrealizowany projekt pokazał, że system AutoML oparty na predefiniowanych konfiguracjach i metodach zespołowych może przynosić satysfakcjonujące wyniki. Warto również podkreślić rolę różnorodności - to właśnie ensemblingi, stworzone na podstawie różniących się algorytmów, często okazywały się modelami, które były wybierane jako najlepsze.

Dodatkowe materiały

Szczegółowy kod źródłowy znajduje się w repozytorium github [GIT].

Literatura

[AG] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, A. Smola, *AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data*, arXiv preprint arXiv:2003.06505, 2020.

[MEM] M. Trofimov et al., *Memento: A Web of Models for Automating Machine Learning*, arXiv preprint arXiv:1908.06846, 2019.

[GIT] Repozytorium github projektu: `https://github.com/kubawini/AutoML-Project-2`

Dodatek: Dokładne konfiguracje modeli bazowych

Model	n_est	Depth	Min Samples	LR	Subsample
gbm_base	100	3	1	0.1000	1.000
gbm1	492	5	8	0.0080	0.591
gbm2	159	5	9	0.0319	0.550
gbm3	184	5	12	0.0347	0.654
gbm4	389	5	10	0.0171	0.565
gbm5	579	4	22	0.0949	0.384
gbm6	596	5	18	0.0619	0.324
gbm7	389	5	10	0.0171	0.565
gbm8	494	5	17	0.0223	0.650
gbm9	382	5	8	0.0230	0.644
gbm10	403	3	25	0.0207	0.324
gbm11	267	1	15	0.0077	0.304
gbm12	368	1	3	0.0083	0.290

Model	Iter/Est	Depth	LR	L2 Reg	Bagging
cat_base	1000	6	0.0300	3.000	1.000
cat1	339	8	0.9221	4.013	1.924
cat2	221	8	0.0703	7.309	1.065
cat3	447	8	0.0408	5.745	2.314
cat4	107	7	0.0018	8.761	1.114
cat5	791	7	0.0180	6.293	2.909
cat6	517	6	0.0268	6.694	0.178
cat7	342	8	0.0253	1.714	0.511
cat8	306	10	0.0662	7.867	1.792
cat9	126	6	0.0507	0.999	2.347
cat10	316	6	0.0304	0.001	0.536

Model	n_est	LR	Depth	Subsample	Min Child
xgboost1	355	0.0324	8	0.868	2.189
xgboost2	184	0.0617	7	0.683	2.566
xgboost3	282	0.0963	14	0.967	1.077
xgboost4	554	0.1727	15	0.966	2.061
xgboost5	421	0.0502	13	0.947	2.336
xgboost6	166	0.0835	13	0.643	1.228

Model	Algorytm	n_est	Min Samples Leaf	Bootstrap
rf_base	RF	100	-	True
rf_score	RF	255	4	True
rf_extra1	ExtraTrees	376	2	True
rf_extra2	ExtraTrees	310	4	False
rf_extra3	ExtraTrees	266	3	False
rf_extra4	ExtraTrees	163	4	False

Model	Penalty	L1 Ratio	C
lr1	elasticnet	0.749	31.21
lr2	elasticnet	0.828	34.64
lr3	elasticnet	0.514	19.11
lr4	elasticnet	0.458	2.97
lr5	elasticnet	0.483	1.83
lr6	elasticnet	0.930	7.69

Model	n_neighbors
knn_3neighbors	3
knn_5neighbors	5
knn_7neighbors	7
knn_10neighbors	10