POLITECHNIKA BIAŁOSTOCKA

WYDZIAŁ MECHANICZNY

PRACOWNIA SPECJALISTYCZNA

Sztuczne sieci neuronowe i systemy ekspertowe

Ćwiczenie nr 3

Projektowanie optymalnego zespołu klasyfikatorów

KOD PRZEDMIOTU: MYAR2S22003M

Autor: Ostaszewicz Dawid

Kierunek: Automatyka i Robotyka, II stopień

Prowadzący: dr inż. Marcin Derlatka

Cel ćwiczenia

Celem ćwiczenia jest zaprojektowanie i implementacja optymalnego zespołu klasyfikatorów dla przesłanego zbioru danych. Zespół ma obejmować co najmniej jeden klasyfikator realizujący każdą z technik: bagging, boosting oraz stacking. Zadanie obejmuje:

- Wykorzystanie klasyfikatorów bazowych opartych na sieciach neuronowych, takich jak MLP (Multilayer Perceptron).
- Implementację klasyfikatorów bagging i boosting, takich jak BaggingClassifier, AdaBoostClassifier oraz GradientBoostingClassifier.
- Zastosowanie metody stacking, która łączy wyniki klasyfikatorów bazowych przy pomocy modelu meta-learnera, np. XGBoost.
- Eksperymentowanie z różnymi hiperparametrami klasyfikatorów oraz technikami optymalizacji, aby uzyskać jak najlepsze wyniki klasyfikacji.

1 Wczytanie danych

Dane wejściowe w formacie ARFF są wczytywane przy użyciu funkcji `arff.loadarff`. Zbiory danych są następnie konwertowane na format `pandas DataFrame` dla łatwiejszej manipulacji danymi.

```
1 Dry_Bean_Dataset_data, Dry_Bean_Dataset_meta = arff.loadarff('
    Dry_Bean_Dataset.arff')
2 testing_beans_data, testing_beans_meta = arff.loadarff('
    testing_beans.arff')
3 training_beans_data, training_beans_meta = arff.loadarff('
    training_beans.arff')
4
5 Dry_Bean_Dataset = pd.DataFrame(Dry_Bean_Dataset_data)
6 testing_beans = pd.DataFrame(testing_beans_data)
7 training_beans = pd.DataFrame(training_beans_data)
```

2 Przygotowanie danych

W tej części skryptu wydzielamy cechy (`X`) i etykiety (`y`) zarówno z danych treningowych, jak i testowych. Etykiety są konwertowane z typu `byte` na `string`.

```
X_train = training_beans.drop('Class', axis=1)
y_train = training_beans['Class']
X_test = testing_beans.drop('Class', axis=1)
y_test = testing_beans['Class']

y_train = y_train.str.decode('utf-8')
y_test = y_test.str.decode('utf-8')
```

3 Kodowanie etykiet

Użyto `LabelEncoder` do kodowania etykiet w postaci liczbowej, co jest wymagane do trenowania klasyfikatorów.

```
le = LabelEncoder()

y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)
```

4 Inicjalizacja klasyfikatorów

Inicjalizowane są klasyfikatory bazowe: sieć neuronowa (`MLPClassifier`) oraz klasyfikator XGBoost (`XGBClassifier`). Sieć neuronowa jest używana w zespole bagging.

```
base_nn = MLPClassifier(hidden_layer_sizes=(100,), max_iter
    =500, random_state=42)
xgb_clf = XGBClassifier(random_state=42)
```

5 Bagging Classifier

Tworzy się klasyfikator `BaggingClassifier`, który stosuje metodę bagging z wykorzystaniem sieci neuronowej jako klasyfikatora bazowego. Klasyfikator jest trenowany na danych treningowych, a następnie dokonujemy predykcji na danych testowych.

```
bagging_clf = BaggingClassifier(estimator=base_nn, n_estimators
    =10, random_state=42)
bagging_clf.fit(X_train, y_train)
```

```
y_pred_bagging = bagging_clf.predict(X_test)
```

6 Boosting Classifier

Następnie tworzymy klasyfikator boostingowy `GradientBoostingClassifier`, który jest trenowany na danych treningowych.

```
boosting_clf = GradientBoostingClassifier(n_estimators=50,
    random_state=42)
boosting_clf.fit(X_train, y_train)
y_pred_boosting = boosting_clf.predict(X_test)
```

7 Stacking Classifier

W przedstawionym fragmencie kodu zastosowano metodę stacking do łączenia predykcji trzech klasyfikatorów bazowych. Proces rozpoczyna się od zdefiniowania trzech modeli bazowych: bagging_clf, boosting_clf oraz xgb_clf, które odpowiadają za generowanie predykcji na podstawie danych treningowych. Następnie, przy użyciu funkcji stacking, wykonywane jest połączenie wyników predykcji tych klasyfikatorów. Dla każdego modelu bazowego obliczane są predykcje "out-of-fold", które stanowią dane wejściowe do meta-learnera.

Funkcja stacking przyjmuje następujące parametry: dane treningowe X_train, y_train, dane testowe X_test, oraz liczbę foldów kroswalidacji (n_folds=4). Ustawienie regression=False wskazuje, że problem jest klasyfikacyjny, a parametr metric=accuracy_score określa miarę oceny modelu. Dodatkowo, parametr stratified=True zapewnia zachowanie proporcji klas w trakcie kroswalidacji.

```
models = [bagging_clf, boosting_clf, xgb_clf]
S_train, S_test = stacking(models,
X_train, y_train, X_test,
regression=False,
mode='oof_pred_bag',
needs_proba=False,
save_dir=None,
metric=accuracy_score,
n_folds=4,
stratified=True,
```

```
    shuffle=True,
    random_state=42,
    verbose=2)

    final_estimator = XGBClassifier(random_state=42)

    final_estimator = final_estimator.fit(S_train, y_train)
    y_pred_stacking = final_estimator.predict(S_test)
```

8 Optymalizacja hiperparametrów dla MLPClassifier

Poniższy kod przedstawia konfigurację siatki hiperparametrów (`param_grid`) do wykorzystania z klasą `GridSearchCV`. Celem jest optymalizacja klasyfikatora `MLPClassifier` z różnymi ustawieniami warstw ukrytych, funkcji aktywacji, solverów oraz parametru regularyzacji `alpha`.

```
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (150,)],
    'activation': ['relu', 'tanh', 'logistic'],
    'solver': ['adam', 'sgd'],
    'alpha': [0.0001, 0.001, 0.01]
}
```

Program automatycznie sprawdza konfiguracje modeli dla parametrów zadanych w tym listingu. Tabela z wynikami została wstawiona na końcu sprawozdania.

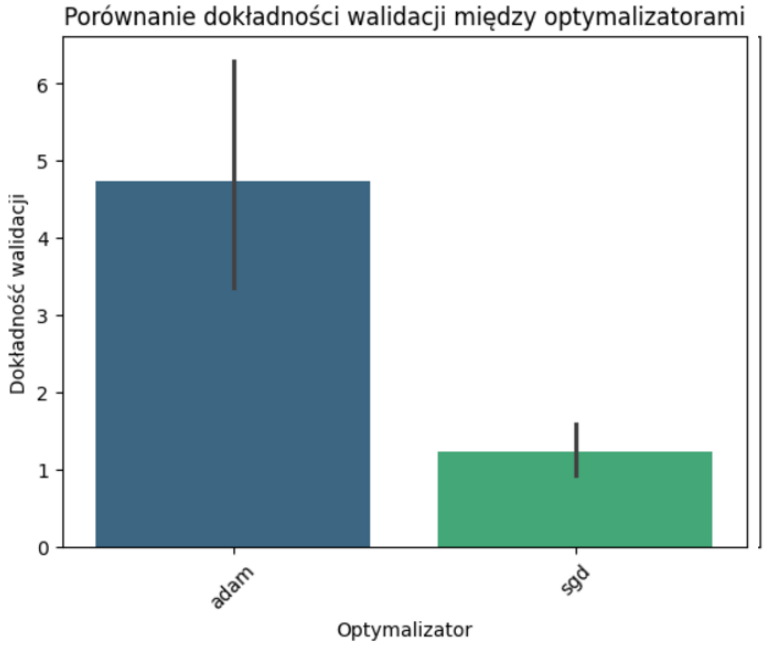
9 Wyniki

Rysunek 1 przedstawia porównanie dokładności walidacji uzyskanej przez modele optymalizatorów: Adam i SGD. Na osi X umieszczone zostały poszczególne optymalizatory, a oś Y przedstawia średnią dokładność walidacji, którą osiągnęły modele w trakcie procesu uczenia. Wysokość słupków pokazuje skuteczność poszczególnych optymalizatorów w poprawie wyników walidacji, umożliwiając ich bezpośrednie porównanie.

Z wykresu wynika, że optymalizator Adam osiąga wyraźnie lepsze wyniki niż SGD. Adam przewyższa SGD pod względem średniej dokładności walidacji, co sugeruje jego większą efektywność w optymalizacji modelu. Powodem, dla którego Adam wypada lepiej, jest jego zdolność do dynamicznego dostosowywania współczynnika uczenia do każdej wagi w modelu. Algorytm

ten korzysta z adaptacyjnych współczynników uczenia, które automatycznie modyfikują tempo nauki na podstawie historii gradientów, co pozwala na szybszą i bardziej stabilną zbieżność, szczególnie w przypadku trudnych funkcji kosztu.

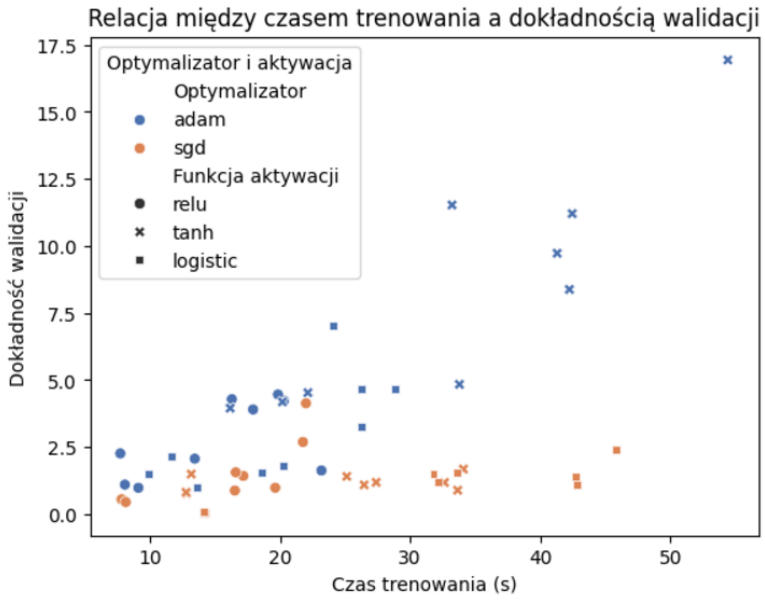
Z kolei optymalizator SGD, mimo że jest prosty i szeroko stosowany, może wykazywać wolniejszą zbieżność i być bardziej podatny na wpadanie w minima lokalne, szczególnie gdy tempo uczenia nie jest odpowiednio dobrane. W związku z tym, dla omawianych modeli, Adam okazuje się bardziej efektywny, co znajduje odzwierciedlenie w wyższej średniej dokładności walidacji w porównaniu do SGD. Rysunek 2 typu scatterplot ilustruje zależność między czasem trenowania



Rysunek 1: Czas treningu

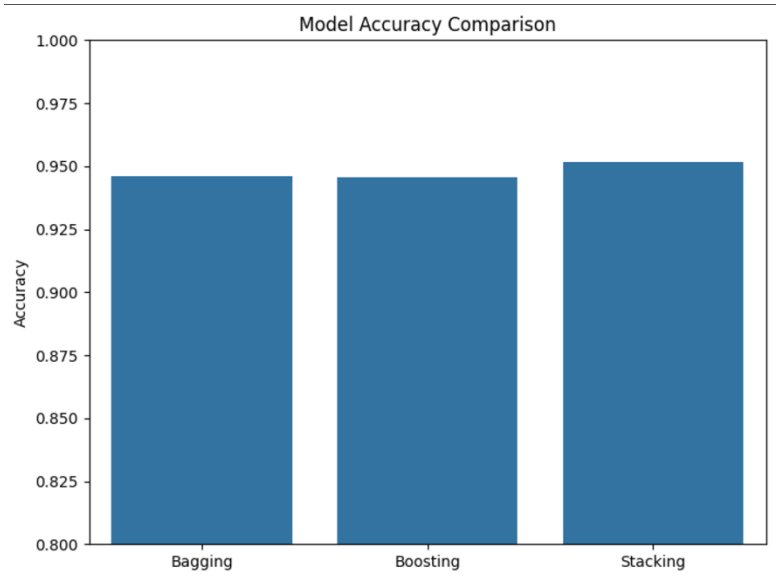
modelu (na osi X) a jego dokładnością walidacji (na osi Y). Punkty na wykresie reprezentują różne optymalizatory i funkcje aktywacji, a ich rozmieszczenie wskazuje, jak czas treningu wpływa na osiąganą dokładność. Każdy punkt jest oznaczony różnym kolorem w zależności od używanego optymalizatora, co pozwala na rozróżnienie wpływu różnych optymalizatorów na wyniki. Dodatkowo, styl punktów wskazuje, która funkcja aktywacji została zastosowana w modelu, co umożliwia ocenę, jak ta zmienna również wpływa na wydajność modeli. Wykres zawiera legendę, która wskazuje, który kolor odpowiada któremu optymalizatorowi oraz jaką funkcję aktywacji reprezentuje dany styl punktu. Dzięki temu wykresowi można szybko zobaczyć, czy istnieje korelacja między czasem trenowania a dokładnością, a także jak różne optymalizatory i funkcje aktywacji wpływają na efektywność modeli.

Ostatecznie, porównanie wyników uzyskanych przez różne metody jest przedstawione na Rysunku 4. Zgodnie z wykresem, metoda stacking osiąga średnio lepsze wyniki w porównaniu do baggingu i boosting. Stacking, który polega na łączeniu predykcji różnych klasyfikatorów



Rysunek 2: Czas treningu

bazowych przy użyciu modelu drugiego poziomu (meta-learnera), wydaje się skuteczniejszy w wykorzystywaniu różnorodności wyników uzyskanych przez modele bazowe.



Rysunek 3: Czas treningu

10 Wnioski

Porównywanie wyników uzyskanych przez różne modele klasyfikacyjne, bazujących na odmiennych założeniach teoretycznych i strukturach wewnętrznych, jest z metodologicznego punktu widzenia nieadekwatne. Brak pełnej znajomości struktury przestrzeni problemowej, w której operują te modele, stanowi kluczowy czynnik uniemożliwiający rzetelną interpretację wyników.

Niejednorodność struktur wewnętrznych sprawia, że porównanie wyników tych modeli jest

nieadekwatne bez uwzględnienia potencjalnych różnic w przestrzeni cech. Przestrzeń problemowa, w której modele te operują, jest często nieliniowa, skomplikowana i zależna od specyfiki zbioru danych. Bez pełnej wiedzy o jej strukturze, nawet dokładne wyniki jednego modelu mogą być nieodpowiednie lub źle zinterpretowane w kontekście innego, co prowadzi do błędnych wniosków.

W związku z tym, brak pełnej świadomości tego, jak dany algorytm przetwarza dane, w jaki sposób interpretuje zależności pomiędzy cechami, a także w jaki sposób przebiega optymalizacja, sprawia, że próby porównań pomiędzy algorytmami mogą prowadzić do mylnych wniosków, które zniekształcają rzeczywisty obraz efektywności modeli w kontekście złożoności problemu.

Jeżeli proces, nawet skomplikowany, można opisać analitycznie, zdecydowanie lepiej jest zastosować podejście analityczne. Każdy z modeli może również różnie reagować na różne dane wejściowe, co nie zostało sprawdzone, dlatego z przeprowadzonych badań nie można wyciągnąć poprawnych wniosków. Przeprowadzenie większej liczby badań na sprawdzenie skuteczności działania modeli, czasowo wykracza poza czas przeznaczony na realizację zajęć. (mam nadzieję, że to nic złego że próbowałem Pythona zamiast Weki, ale ciekawiło mnie jak to by działa :).

Ogólnie czas uzyskiwania tabelki na końcu to trochę ponad 2 godziny. Ciekawe jest też to, że można było wygenerować dane bezpośrednio do formuły Latex z poziomu Pythona. Tylko się nie mieściła na stornie, ale można ją przybliżyć w pdf i nie powinna tracić jakości.

Czas trenowania (s)	Dokładność walidacji	Czas predykcji (s)	Średni błąd na epokę	Funkcja aktywacji	Regularyzacja	Warstwy ukryte	Optymalizator	Słownik hiperparametrów	Średnia dokładność CV	Wynik CV1	Wynik CV2	Wynik CV3	Wynik CV4	Wynik CV5	Odchylenie CV	Liczba epok
9.416931	0.987091	0.005035	0.002176	relu	0.0001000	(50,)	adam	'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'solver': 'adam'	0.927072	0.928122	0.932844	0.935433	0.929659	0.930626	0.003094	13
7.944553	0.549052	0.007089	0.003458	relu	0.0001000	(50,)	sgd	'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'solver': 'sgd'	0.910283	0.922875	0.928646	0.927034	0.923360	0.922440	0.006457	43
16.299733	4.289404	0.005496	0.000175	relu	0.0001000	(100,)	adam	'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'adam'	0.920776	0.926023	0.934942	0.936483	0.930709	0.929787	0.005790	17
17.818887	1.431041	0.005480	0.000147	relu	0.0001000	(100,)	sgd	'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'sgd'	0.910283	0.922350	0.928646	0.924409	0.929659	0.923670	0.006033	37
20.201666	4.237675	0.006403	0.000170	relu	0.0001000	(150,)	adam	'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (150,), 'solver': 'adam'	0.922350	0.926023	0.935992	0.937408	0.929699	0.929066	0.005679	16
21.765174	2.694443	0.006174	0.000113	relu	0.0001000	(150,)	sgd	'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (150,), 'solver': 'sgd'	0.912382	0.923400	0.930220	0.927034	0.929084	0.924224	0.006319	29
7.729759	2.267514	0.005387	0.002062	relu	0.0001000	(50,)	adam	'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'solver': 'adam'	0.925498	0.929696	0.935992	0.932808	0.931234	0.931046	0.003470	8
7.828698	0.556461	0.005655	0.002354	relu	0.0001000	(50,)	sgd	'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'solver': 'sgd'	0.912907	0.919727	0.927072	0.928084	0.925459	0.922650	0.005665	40
19.840223	4.465088	0.006350	0.001851	relu	0.0001000	(100,)	adam	'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'solver': 'adam'	0.925498	0.928122	0.935467	0.935483	0.931234	0.931151	0.003052	7
16.533072	0.885784	0.007312	0.003830	relu	0.0001000	(100,)	sgd	'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'solver': 'sgd'	0.911857	0.924449	0.928122	0.924934	0.923084	0.923069	0.005747	38
23.183464	1.633910	0.007606	0.002930	relu	0.0001000	(150,)	adam	'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (150,), 'solver': 'adam'	0.923924	0.924449	0.937566	0.937008	0.930184	0.930626	0.005868	12
21.097824	4.137875	0.007313	0.002523	relu	0.0001000	(150,)	sgd	'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (150,), 'solver': 'sgd'	0.911333	0.922875	0.928122	0.925459	0.929134	0.923385	0.006408	35
8.081057	1.102558	0.005721	0.002942	relu	0.0010000	(50,)	adam	'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50,), 'solver': 'adam'	0.923400	0.929171	0.933893	0.934383	0.930709	0.930311	0.003066	15
8.146571	0.456436	0.004251	0.001134	relu	0.0010000	(50,)	sgd	'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (50,), 'solver': 'sgd'	0.915005	0.923924	0.927597	0.927034	0.930184	0.924749	0.005263	28
13.459869	2.078855	0.006647	0.001591	relu	0.0010000	(100,)	adam	'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'solver': 'adam'	0.923250	0.929696	0.935467	0.938583	0.934908	0.932201	0.005603	2
16.660931	1.563143	0.005543	0.000264	relu	0.0010000	(100,)	sgd	'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'solver': 'sgd'	0.912907	0.923924	0.929171	0.928084	0.925459	0.923909	0.005896	30
17.932832	3.906863	0.007166	0.002102	relu	0.0010000	(150,)	adam	'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (150,), 'solver': 'adam'	0.923924	0.928646	0.941238	0.938583	0.930709	0.932620	0.004601	1
19.635134	0.989184	0.006174	0.000076	relu	0.0010000	(150,)	sgd	'activation': 'relu', 'alpha': 0.01, 'hidden_layer_sizes': (150,), 'solver': 'sgd'	0.914481	0.921826	0.926548	0.924934	0.929659	0.923489	0.005166	33
20.191227	4.188578	0.011041	0.005528	tanh	0.0000100	(50,)	adam	'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'solver': 'adam'	0.921301	0.926023	0.938090	0.939633	0.931759	0.931361	0.006079	6
12.813363	0.768418	0.013211	0.004045	tanh	0.0000100	(50,)	sgd	'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'solver': 'sgd'	0.911857	0.920262	0.928122	0.923360	0.927559	0.922240	0.002931	45
41.296467	9.728540	0.018344	0.002622	tanh	0.0000100	(100,)	adam	'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'adam'	0.922350	0.924449	0.935992	0.937008	0.932808	0.931021	0.004014	14
26.467898	1.094161	0.025116	0.008408	tanh	0.0000100	(100,)	sgd	'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'sgd'	0.912382	0.920776	0.926548	0.926509	0.926509	0.922545	0.005547	42
54.413109	16.933176	0.023445	0.000221	tanh	0.0000100	(150,)	adam	'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (150,), 'solver': 'adam'	0.924974	0.924449	0.935992	0.931759	0.930709	0.929576	0.004351	18
34.096626	1.684655	0.027096	0.008859	tanh	0.0000100	(150,)	sgd	'activation': 'tanh', 'alpha': 0.0001, 'hidden_laye_sizes': (150,), 'solver': 'sgd'	0.912382	0.921826	0.927597	0.925984	0.929699	0.923280	0.005020	36
22.143486	4.523276	0.010608	0.000535	tanh	0.0001000	(50,)	adam	'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'solver': 'adam'	0.921826	0.927597	0.938615	0.937008	0.933958	0.932201	0.004437	3
13.418641	1.468212	0.011049	0.000062	tanh	0.0001000	(50,)	sgd	'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'solver': 'sgd'	0.911857	0.924674	0.929171	0.925984	0.929699	0.923699	0.006081	32
42.239000	8.380962	0.020772	0.007589	tanh	0.0001000	(100,)	adam	'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'solver': 'adam'	0.921826	0.924974	0.937566	0.935958	0.934383	0.930941	0.006318	9
27.391706	1.189950	0.020367	0.006422	tanh	0.0001000	(100,)	sgd	'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'solver': 'sgd'	0.912382	0.921826	0.928122	0.924409	0.924934	0.922335	0.005364	44
33.217559	11.527404	0.029478	0.008935	tanh	0.0001000	(150,)	adam	'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (150,), 'solver': 'adam'	0.922875	0.928122	0.934942	0.935958	0.932808	0.930941	0.004851	10
32.643888	1.184471	0.029667	0.008614	tanh	0.0001000	(150,)	sgd	'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (150,), 'solver': 'sgd'	0.913431	0.921826	0.928646	0.926509	0.929084	0.923099	0.005666	31
16.474351	3.056814	0.010153	0.000267	tanh	0.0010000	(50,)	adam	'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (50,), 'solver': 'adam'	0.921301	0.928646	0.937566	0.935433	0.931759	0.930941	0.005708	11
12.78585	0.820891	0.015218	0.006098	tanh	0.0010000	(50,)	sgd	'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (50,), 'solver': 'sgd'	0.912382	0.919727	0.926023	0.924409	0.930184	0.922545	0.006085	41
33.789020	4.842609	0.017818	0.001766	tanh	0.0010000	(100,)	adam	'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'solver': 'adam'	0.922875	0.926023	0.934942	0.940682	0.933858	0.931676	0.004418	5
25.534328	1.411759	0.017434	0.001328	tanh	0.0010000	(100,)	sgd	'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'solver': 'sgd'	0.911857	0.920776	0.927072	0.927034	0.930184	0.923385	0.006524	34
42.452232	11.216068	0.024035	0.001014	tanh	0.0010000	(150,)	adam	'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (150,), 'solver': 'adam'	0.922875	0.927597	0.940189	0.938058	0.932283	0.932310	0.006423	4
33.653533	0.899066	0.028084	0.008364	tanh	0.0010000	(150,)	sgd	'activation': 'tanh', 'alpha': 0.01, 'hidden_layer_sizes': (150,), 'solver': 'sgd'	0.913333	0.921301	0.927597	0.924934	0.929659	0.922965	0.006453	39
13.611701	1.011785	0.006149	0.000495	logistic	0.0000100	(50,)	adam	'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'solver': 'adam'	0.919727	0.923924	0.935992	0.934908	0.930709	0.929052	0.006300	23
14.140672	0.655523	0.006248	0.000342	logistic	0.0000100	(50,)	sgd	'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (50,), 'solver': 'sgd'	0.912382	0.924449	0.921260	0.920735	0.918242	0.921960	0.002052	52
26.273873	4.652889	0.008690	0.000178	logistic	0.0000100	(100,)	adam	'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'adam'	0.919203	0.924674	0.933368	0.935958	0.931759	0.929052	0.006120	22
33.623173	1.556593	0.008560	0.000100	logistic	0.0000100	(100,)	sgd	'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'sgd'	0.909759	0.913431	0.925498	0.922310	0.924409	0.919082	0.006306	49
24.095789	7.029436	0.013119	0.004144	logistic	0.0000100	(150,)	adam	'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (150,), 'solver': 'adam'	0.917104	0.929646	0.933368	0.936483	0.930184	0.929157	0.006603	20
42.725519	1.461274	0.010993	0.000115	logistic	0.0000100	(150,)	sgd	'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (150,), 'solver': 'sgd'	0.911857	0.912907	0.924674	0.922310	0.925459	0.919501	0.005020	48
11.695150	2.167251	0.009483	0.004443	logistic	0.0001000	(50,)	adam	'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'solver': 'adam'	0.923924	0.926023	0.935467	0.932808	0.929084	0.929261	0.004277	19
14.300560	0.957058	0.011265	0.002984	logistic	0.0001000	(50,)	sgd	'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (50,), 'solver': 'sgd'	0.909234	0.911857	0.922350	0.923360	0.921260	0.917612	0.005867	53
26.299041	3.263118	0.009384	0.001437	logistic	0.0001000	(100,)	adam	'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'solver': 'adam'	0.921301	0.924449	0.934418	0.934383	0.931234	0.929157	0.003551	21
32.156416	1.222405	0.010985	0.002384	logistic	0.0001000	(100,)	sgd	'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'solver': 'sgd'	0.911857	0.912382	0.923400	0.921785	0.923360	0.918557	0.003291	51
28.806508	4.652324	0.011008	0.000092	logistic	0.0001000	(150,)	adam	'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (150,), 'solver': 'adam'	0.921301	0.922875	0.935516	0.934908	0.929659	0.929052	0.006142	24
42.846591	1.093333	0.011608	0.000979	logistic	0.0001000	(150,)	sgd	'activation': 'logistic', 'alpha': 0.001, 'hidden_layer_sizes': (150,), 'solver': 'sgd'	0.912907	0.926548	0.924409	0.926509	0.920751	0.006504	46
9.879049	1.509752	0.008536	0.003271	logistic	0.0010000	(50,)	adam	'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (50,), 'solver': 'adam'	0.917104	0.925498	0.932844	0.934908	0.930184	0.928508	0.003338	25
14.417625	0.102597	0.011220	0.003520	logistic	0.0010000	(50,)	sgd	'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (50,), 'solver': 'sgd'	0.909759	0.922350	0.922310	0.920210	0.916773	0.920394	0.002394	54
18.555469	1.546875	0.010227	0.002896	logistic	0.0010000	(100,)	adam	'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'solver': 'adam'	0.919727	0.922875	0.930745	0.933333	0.932283	0.927703	0.005456	26
31.792329	1.507184	0.009653	0.001767	logistic	0.0010000	(100,)	sgd	'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (100,), 'solver': 'sgd'	0.911431	0.913431	0.925498	0.923360	0.921785	0.919081	0.005635	50
20.254692	1.796257	0.013939	0.006020	logistic	0.0010000	(150,)	adam	'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (150,), 'solver': 'adam'	0.916579	0.924974	0.930745	0.930184	0.931234	0.926743	0.005558	27
45.856850	2.402260	0.017029	0.005925	logistic	0.0010000	(150,)	sgd	'activation': 'logistic', 'alpha': 0.01, 'hidden_layer_sizes': (150,), 'solver': 'sgd'	0.913431	0.913431	0.926023	0.923360	0.924409	0.920131	0.005536	47