Inteligencja Obliczeniowa

Praca domowa nr 4 – Przetwarzanie obrazu

Krzysztof Kulewski, 238149, grupa 1, 19.01.2019

1. Opis zadania

Celem zadania było dokonanie klasyfikacji obrazów przy użyciu wybranych klasyfikatorów, a następnie zestawienie otrzymanych wyników i ich analiza.

2. Wybrany zbiór danych i obróbka

W projekcie postanowiłem posłużyć zbiorem MNIST, czyli zestawem cyfr od 0 do 9 pisanych odręcznie. W jego skład wchodzi 60 000 obrazów treningowych oraz 10 000 obrazów testowych, które zostały użyte do ewaluacji. Każdy z obrazów ma wielkość 28 x 28 pikseli, natomiast każdy piksel jest reprezentowany przez liczbę w zakresie 0 – 255, która opisuje nasycenie kanału alfa.
Zbiór MNIST zdobył ogromną popularność i stał się wręcz kanoniczny dla zagadnienia klasyfikacji obrazów.

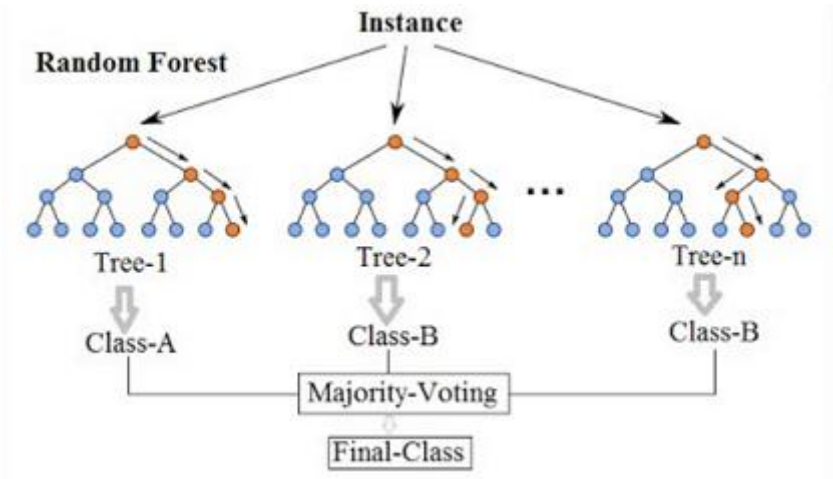
Już na samym początku projektu okazało się, że praca na obrazach w formie plików jest bardzo niewygodna, stąd decyzja, by zamiast plików graficznych, użyć pliku CSV, w którego skład wchodzi 785 kolumn – pierwsza to klasa (cyfry od 0 do 9), a 784 pozostałe reprezentują wartość kanału alfa dla kolejnych pikseli obrazu. Wielkość pliku treningowego to około 110 MB, testowego – 20 MB.

Dane zostały dodatkowo obrobione – wydzielono macierze danych i wektory klas oraz poprawiono nazwy kolumn tak, by mogły być przekazywane do klasyfikatorów:

```
library(readr)
train = read_csv("data/mnist_train.csv")
test = read_csv("data/mnist_test.csv")

train.labels = train[, 1]
train.labels = as.factor(train.labels$label)
test.labels = test[, 1]
test.labels = as.factor(test.labels$label)

train.data = as.matrix(train[, -1])
test.data = as.matrix(test[, -1])

names(train.data) = make.names(names(train.data))
names(test.data) = make.names(names(test.data))
```

3. Klasyfikator I – Random Forest (Lasy Losowe)

Pierwszy z użytych klasyfikatorów to lasy losowe. Lasy losowe to rozszerzenie drzew decyzyjnych, które polegają na schodzeniu od korzenia do liści, a decyzja o wyborze lewego lub prawego dziecka danego węzła zostaje podjęta w oparciu o wartość parametru, który testuje dany węzeł.
W lasach losowych, zamiast jednego drzewa, tworzony jest „las" składający się z wielu różnych drzew. Wynikiem klasyfikacji jest cyfra, która została wybrana przez największą liczbę drzew.



Postanowiłem uruchomić klasyfikator RandomForest z wartościami: 10 drzew, 10 węzłów.
Model został użyty do klasyfikacji danych testowych. Stworzono również macierz błędów:

```
library(randomForest)
model.rf10n10 = randomForest(train.data, train.labels, ntree = 10, maxnodes = 10)
pred.model.rf10n10 = predict(model.rf10n10, test.data, type = "class")
cm.rf10n10 = table(pred.model.rf10n10, test.labels)
acc.rf10n10 = sum(diag(cm.rf10n10))/sum(cm.rf10n10)
```

Macierz błędów RF dla 10 drzew, 10 węzłów:

	test.labels									
.rf10n10	0	1	2	3	4	5	6	7	8	9
0	856	0	11	19	2	34	73	6	28	6
1	3	1080	70	97	11	101	22	70	115	19
2	34	39	834	80	47	81	349	49	249	29
3	14	9	23	657	7	245	22	6	125	11
4	3	1	16	19	711	135	111	37	87	277
5	17	0	4	50	1	130	3	1	6	2
6	10	1	14	2	4	19	222	2	24	2
7	21	4	47	53	107	46	96	831	57	160
8	1	0	4	0	0	9	5	0	153	3
9	21	1	9	33	92	92	55	26	130	500

Dokładność to 59,74%

Wartość taka świadczy o dwóch rzeczach:
- z pewnością nie był to wybór losowy, gdyż w takim przypadku należałoby oczekiwać ok. 10%,
- jest to wynik mizerny, gdyż dla zbioru MNIST istnieją klasyfikatory o dokładności 99,5%+

Sprawdziłem więc, jak zmiana liczby drzew oraz liczby węzłów wpłynie na uzyskane wyniki.

Macierz błędów RF dla 10 drzew, 50 węzłów:

rf10n50 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	917	0	8	11	0	19	27	1	16	8
1	2	1092	19	11	3	20	11	26	17	5
2	1	4	859	42	8	7	18	37	25	7
3	2	7	25	786	5	166	12	7	47	23
4	3	0	16	6	755	29	36	22	17	36
5	15	0	11	53	2	491	28	1	16	10
6	17	2	17	3	24	25	776	5	14	12
7	4	1	20	16	15	11	7	827	12	34
8	16	28	50	50	45	79	37	29	788	26
9	3	1	7	32	125	45	6	73	22	848

Dokładność to 81,39%

Macierz błędów RF dla 50 drzew, 10 węzłów:

rf50n10 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	962	0	38	39	6	65	57	6	24	10
1	2	1122	175	69	21	70	42	62	93	26
2	2	6	679	34	10	8	19	27	45	5
3	5	3	33	785	9	370	7	0	76	21
4	0	0	15	15	822	75	28	19	52	315
5	0	0	0	1	2	120	3	0	1	0
6	5	2	25	8	24	48	771	0	54	8
7	2	1	48	32	33	27	17	904	25	119
8	2	1	19	14	4	52	11	2	571	8
9	0	0	0	13	51	57	3	8	33	497

Dokładność to 72,33%.

Jak widać, zwiększenie liczby węzłów znacznie lepiej wpłynęło na jakość klasyfikacji.
RF50N10 to liczny las, w którym poszczególne drzewa mają relatywnie niską dokładność.
RF10N50 to natomiast mniejszy las, ale z dużo dokładniejszymi drzewami. Możemy spodziewać się, że w takim przypadku werdykt był bardziej jednomyślny. Warto odnotować również fakt, że cyfra „5” jest zdecydowanie najtrudniejsza w klasyfikacji, szczególnie dla lasów o małej liczbie węzłów.

Ostatnia kombinacja miała na celu sprawdzenie jak bardzo jesteśmy w stanie polepszyć dokładność przez dalsze zwiększanie parametrów.

Macierz błędów RF dla 500 drzew, 500 węzłów:

rf500n500 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	967	0	7	2	2	6	10	1	4	6
1	1	1119	0	0	1	5	3	8	2	7
2	0	3	961	20	3	1	1	25	6	1
3	0	2	10	930	0	20	0	2	9	13
4	0	0	13	2	892	5	9	6	7	17
5	2	1	1	18	1	820	9	0	6	4
6	4	4	8	0	9	11	921	0	8	1
7	1	1	16	15	0	4	0	943	5	5
8	4	5	11	13	11	9	5	4	901	13
9	1	0	5	10	63	11	0	39	26	942

Dokładność to 93,96%

Uzyskany wynik wydaje się być całkiem dobry. Tak jak poprzednio, największe problemy sprawia klasyfikacja cyfry „5”. Ciekawą obserwacją jest również to, iż cyfra „4” była często uznawana za „9”.

4. Klasyfikator II – Gradient Boosted Trees

Podobnie jak w przypadku pierwszego klasyfikatora, także i tutaj mówimy o rozszerzeniu drzew decyzyjnych. Technika gradient boosting polega na stworzeniu grupy „słabych", prostych klasyfikatorów (analizujących podzbiór parametrów), a następnie iteracyjnym łączeniu ich w całość, celem uzyskania jednego, dokładnego klasyfikatora.
Klasyfikator ten pozwala na manipulację kilkoma parametrami, między innymi:
- n.trees - liczba prymitywnych klasyfikatorów (prostych drzew)
- interaction.depth – liczba podziałów przypadających na węzeł

Oto kod odpowiadający za wygenerowanie modelu, klasyfikację danych testowych oraz stworzenie macierzy błędów:

```
library(gbm)
model.gbt10i2 = gbm.fit(train.data,  train.labels, distribution="multinomial", n.trees=10,
interaction.depth=2)
pred.model.gbt10i2 = predict(model.gbt10i2, test.data, n.trees=model.gbt10i2$n.trees)
cm.gbt10i2 = table(pred.model.gbt10i2, test.labels)
acc.gbt10i2 = sum(diag(cm.gbt10i2))/sum(cm.gbt10i2)
```

Macierz błędów GBT dla 10 drzew i 2 podziałów na węzeł:

gbt10i2	test.labels 0	1	2	3	4	5	6	7	8	9
0	603	0	23	11	1	10	34	26	6	7
1	2	994	110	39	14	21	53	14	215	17
2	193	39	463	78	135	152	54	40	54	133
3	10	19	41	711	21	234	21	13	88	26
4	34	26	86	14	668	90	140	61	29	33
5	26	0	0	0	3	265	7	1	9	0
6	25	8	190	18	10	21	593	0	9	0
7	1	0	1	32	16	25	0	522	15	228
8	84	49	115	72	71	49	56	39	513	93
9	2	0	3	35	43	25	0	312	36	472

Dokładność to 58,04%

Podobnie jak w przypadku klasyfikatora Random Forest, uzyskana dokładność jest kiepska, ale z pewnością nie jest to wybór losowy. Warto więc sprawdzić jak zwiększanie parametrów wpłynie na uzyskane wyniki.

Macierz błędów GBT dla 10 drzew i 8 podziałów na węzeł:

gbt10i8 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	895	0	57	18	30	34	22	17	13	12
1	2	1030	17	9	7	11	6	11	28	5
2	8	19	721	35	18	5	10	17	18	19
3	4	45	76	802	50	80	6	21	60	30
4	5	1	19	4	673	23	41	10	20	17
5	20	6	10	33	51	627	37	12	38	29
6	4	9	30	5	8	22	808	0	7	0
7	1	0	4	11	6	11	0	815	10	92
8	40	24	96	38	75	39	28	21	738	33
9	1	1	2	55	64	40	0	104	42	772

Dokładność to 78,81%

Macierz błędów GBT dla 50 drzew i 2 podziałów na węzeł:

gbt50i2 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	629	0	26	11	1	10	32	27	7	7
1	1	1035	112	54	20	20	28	17	115	20
2	170	21	463	66	124	137	53	38	50	125
3	9	7	33	723	21	249	17	13	88	27
4	33	21	76	14	672	92	161	57	29	30
5	26	0	0	0	3	265	7	1	9	0
6	24	8	191	20	10	22	599	0	9	0
7	1	0	1	11	15	14	0	512	4	219
8	85	43	127	58	66	45	61	15	605	46
9	2	0	3	53	50	38	0	348	58	535

Dokładność to 60,38%

Nie ulega żadnym wątpliwościom to, że zwiększenie parametru interaction.depth ma dużo większy wpływ na uzyskaną dokładność, która wzrosła z 58% do 78%. Pięciokrotne zwiększenie liczby drzew miało natomiast marginalny wpływ na dokładność, polepszając ją zaledwie dwa punkty procentowe. Bardzo ciekawe wydaje się to, że GBT popełnia całkowicie inne błędy niż Random Forest. Znacznie lepiej rozpoznawane są cyfry „5” i „9”, natomiast dużo gorzej - „1”.

Oto wynik zwiększenia obu parametrów, analogicznie do pierwszego klasyfikatora.

Macierz błędów GBT dla 50 drzew i 8 podziałów na węzeł:

gbt50i8 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	894	1	38	22	22	33	20	17	7	12
1	3	1067	23	4	13	11	3	10	37	8
2	8	13	794	40	9	8	5	13	10	14
3	4	18	43	806	32	66	3	18	61	25
4	5	1	24	7	686	22	30	15	16	16
5	13	2	7	31	66	643	41	11	44	28
6	7	8	28	5	9	24	828	0	6	0
7	3	0	5	11	7	13	0	819	10	85
8	43	24	69	35	75	33	28	16	744	33
9	0	1	1	49	63	39	0	109	39	788

Dokładność to 80,69%

Tym razem, strategia ta okazała się być dużo mniej efektywna – wynik jest bardzo podobny do tego, który uzyskano dla pięciokrotnie mniejszej liczby drzew. Wygląda więc na to, że warto wykonać jeszcze jeden test, w którym znacznie zwiększony zostanie parametr interaction.depth.

Macierz błędów GBT dla 10 drzew i 40 podziałów na węzeł:

	test.labels									
gbt10i40	0	1	2	3	4	5	6	7	8	9
0	957	1	14	7	4	12	11	4	7	7
1	1	1104	7	3	3	6	2	5	5	5
2	3	9	913	30	4	3	5	23	16	6
3	0	6	29	893	10	26	2	16	24	21
4	4	1	10	2	883	5	7	4	12	43
5	3	1	1	13	5	779	27	5	10	12
6	5	6	12	5	7	15	877	0	7	5
7	1	0	8	15	6	8	0	908	6	18
8	6	7	37	26	19	19	27	14	869	24
9	0	0	1	16	41	19	0	49	18	868

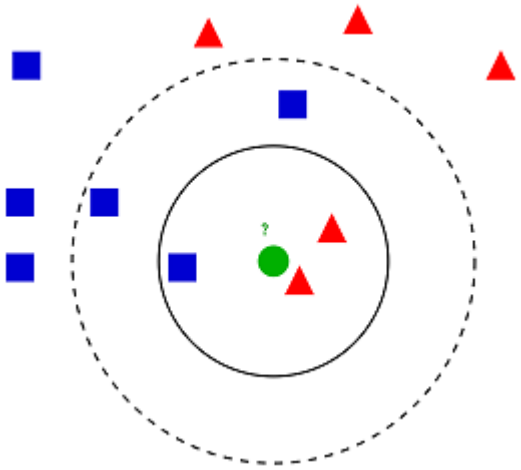
Dokładność to 90,51%

Założenie okazało się słuszne – uzyskana dokładność jest dosyć bliska tej, którą otrzymałem dla Random Forest przy 500 drzewach i 500 węzłach.

Maksymalna wartość parametru interaction.depth to 49, a jego zwiększanie istotnie wpływa na czas obliczeń.

5. Klasyfikator III – K najbliższych sąsiadów (kNN)

W algorytmie K najbliższych sąsiadów, model tworzony jest poprzez porównywanie parametrów zbioru treningowego z jego etykietami. Następnie, dla każdego obiektu (parametrów) w zbiorze testowym, wyszukiwane jest K najbliższych mu sąsiadów, których odległość jest liczona – na ogół – metryką euklidesową. Jest to bardzo prosty klasyfikator.



Najbardziej pasująca wartość parametru K może być odnaleziona stosując automatyczną walidację krzyżową, ale – konsekwentnie względem poprzednich dwóch klasyfikatorów – spróbuję odnaleźć ją ręcznie.

Poniższy kod odpowiada za stworzenie modelu i klasyfikację, oraz generuje macierz błędów:

```
library(FNN)
model.knn5 = knn(train.data, test.data, train.labels, k = 5)
pred.model.knn5 = model.knn5
cm.knn5 = table(pred.model.knn5, test.labels)
acc.knn5 = sum(diag(cm.knn5))/sum(cm.knn5)
```

Macierz błędów kNN dla k = 5:

.knn5 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	974	0	11	0	3	5	5	0	8	5
1	1	1133	8	3	7	0	3	22	3	7
2	1	2	991	3	0	0	0	4	5	3
3	0	0	2	976	0	12	0	0	13	9
4	0	0	1	1	944	2	3	3	6	7
5	1	0	0	13	0	862	2	0	12	3
6	2	0	1	1	4	4	945	0	5	1
7	1	0	15	6	2	1	0	988	5	10
8	0	0	3	3	1	2	0	0	913	2
9	0	0	0	4	21	4	0	11	4	962

Dokładność to 96,88%

Inicjalna dokładność, którą udało się uzyskać, jest najlepsza ze wszystkich badanych dotąd klasyfikatorów i bardzo bliska tej, którą osiąga człowiek – 97,5%.
Może to świadczyć o tym, że klasyfikator kNN świetnie nadaje się do rozpoznawania pisma odręcznego, lub o tym, że parametr k został idealnie „trafiony".
Aby się o tym przekonać, postanowiłem sprawdzić uzyskaną dokładność dla większego i mniejszego parametru k.

Macierz błędów kNN dla k = 3:

.knn3	test.labels 0	1	2	3	4	5	6	7	8	9
0	974	0	10	0	1	6	5	0	8	4
1	1	1133	9	2	6	1	3	21	2	5
2	1	2	996	4	0	0	0	5	4	2
3	0	0	2	976	0	11	0	0	16	8
4	0	0	0	1	950	2	3	1	8	9
5	1	0	0	13	0	859	3	0	11	2
6	2	0	0	1	4	5	944	0	3	1
7	1	0	13	7	2	1	0	991	4	8
8	0	0	2	3	0	3	0	0	914	2
9	0	0	0	3	19	4	0	10	4	968

Dokładność to 97,05%

Macierz błędów kNN dla k = 7:

knn7	test.labels 0	1	2	3	4	5	6	7	8	9
0	974	0	11	0	1	5	6	0	6	5
1	1	1133	8	3	8	0	3	25	4	6
2	1	2	988	2	0	0	0	3	6	3
3	0	0	2	976	0	8	0	0	11	6
4	0	0	1	1	945	2	3	1	7	8
5	1	0	0	12	0	866	2	0	12	4
6	2	0	2	1	5	4	944	0	1	1
7	1	0	16	7	1	1	0	989	6	11
8	0	0	4	4	1	2	0	0	916	2
9	0	0	0	4	21	4	0	10	5	963

Dokładność to 96,94%

Zarówno zmniejszenie, jak i zwiększenie parametru k sprawiło, że uzyskany wynik jest lepszy.
W przypadku k = 3 dokładność poprawiła się o 0,17 punkta procentowego, natomiast k =7 poprawiło początkowy wynik o 0,06 punkta procentowego. Wygląda więc na to, że warto iść krok dalej i sprawdzić zachowanie dla k = 1 i k = 9.

Macierz błędów kNN dla k = 1:

.knn1	test.labels 0	1	2	3	4	5	6	7	8	9
0	973	0	7	0	0	1	4	0	6	2
1	1	1129	6	1	7	1	2	14	1	5
2	1	3	992	2	0	0	0	6	3	1
3	0	0	5	970	0	12	0	2	14	6
4	0	1	1	1	944	2	3	4	5	10
5	1	1	0	19	0	860	5	0	13	5
6	3	1	2	0	3	5	944	0	3	1
7	1	0	16	7	5	1	0	992	4	11
8	0	0	3	7	1	6	0	0	920	1
9	0	0	0	3	22	4	0	10	5	967

Dokładność to 96,91%

Macierz błędów kNN dla k = 9:

knn9 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	972	0	14	0	2	5	6	0	7	6
1	1	1132	13	3	11	0	4	27	4	7
2	1	2	978	1	0	0	0	3	4	2
3	0	0	3	976	0	8	0	0	12	11
4	0	0	1	1	936	2	3	1	6	10
5	2	0	0	11	0	867	2	0	13	3
6	3	1	2	1	5	4	943	0	3	1
7	1	0	16	8	1	1	0	986	8	9
8	0	0	5	4	1	1	0	0	911	2
9	0	0	0	5	26	4	0	11	6	958

Dokładność to 96,59%

Jak widać, dalsze zwiększanie / zmniejszanie parametrów sprawia, że dokładność spada.
Zostały więc jeszcze dwie wartości parametru K, sąsiednie do K = 3, dla którego uzyskano najlepszą dokładność – 2 i 4.

Macierz błędów KNN dla K = 2:

knn2 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	976	0	11	1	3	6	7	0	10	6
1	1	1133	10	1	7	2	3	29	2	6
2	1	2	995	8	0	0	0	8	8	3
3	0	0	1	981	0	25	0	2	28	9
4	0	0	2	1	959	2	5	3	9	19
5	1	0	0	9	0	850	4	0	29	4
6	0	0	0	0	2	2	939	0	4	1
7	1	0	12	6	3	1	0	981	5	22
8	0	0	1	2	0	1	0	0	876	2
9	0	0	0	1	8	3	0	5	3	937

Dokładność to 96,27%

Macierz błędów KNN dla K = 4:

knn4 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	976	0	11	0	3	4	6	0	9	4
1	1	1133	8	2	9	0	3	23	3	6
2	1	2	993	5	0	0	0	5	5	3
3	0	0	1	979	0	17	0	0	16	6
4	0	0	1	1	952	2	4	5	6	13
5	1	0	0	10	0	862	3	0	19	4
6	0	0	0	1	4	5	942	0	5	1
7	1	0	16	7	1	1	0	988	5	16
8	0	0	2	2	0	0	0	0	902	1
9	0	0	0	3	13	1	0	7	4	955

Dokładność to 96,82%

Parametrem, dla którego uzyskano najwyższą dokładność, **97,05%,** pozostaje więc **K = 3**.
Warto odnotować fakt, że dla każdej badanej wartości (od 1 do 9), model uzyskiwał bardzo wysoką dokładność (96%+), więc stosunkowo prosty klasyfikator K najbliższych sąsiadów świetnie sprawdza się w badanym zagadnieniu.

6. Klasyfikator IV – Regresja brzegowa (Ridge regression)

Regresja liniowa polega na takim dopasowaniu funkcji liniowej, by jak najlepiej oddawała charakter danych. Jest to technika, która niezbyt dobrze sprawdza się w problemie klasyfikacji, dlatego zamiast niej, postanowiłem przeanalizować jej rozszerzenie – regresję brzegową.

Główną zaletą regresji brzegowej jest to, iż wprowadza dodatkowy parametr alfa.
Parametr alfa wchodzi w skład otrzymanego równania prostej (lub hiperpłaszczyzny), a jego zadaniem jest zmniejszenie wpływu parametru wolnego (bias – specyficznego dla danej próbki danych) na rzecz regulacji kąta nachylenia prostej (lub hiperpłaszczyzny).
Unikamy dzięki temu sytuacji, gdy otrzymany model jest nadmiernie dopasowany do zbioru treningowego, przez co sprawdzałby się gorzej w klasyfikacji zbioru testowego.

Jako że regresja brzegowa jest dosyć czasochłonna, testy wyszukujące najlepszy zestaw parametrów zostaną przeprowadzone na dużo mniejszym podzbiorze danych treningowych – 1000 elementów.
W przypadku badanego problemu, manualne tworzenie wektorów z wartościami lambda (dla każdej zmiennej) byłoby karkołomne, stąd zadaniem tym zajmie się sam algorytm, ja natomiast będę regulował parametr nfold, który odpowiada za ilość partycji, w obrębie których zostanie przeprowadzona automatyczna walidacja krzyżowa.

Kod odpowiedzialny za stworzenie modelu, klasyfikację i macierz błędów:

```
library(glmnet)
model.rrf10 = cv.glmnet(train.data[1:1000,], train.labels[1:1000], lambda = NULL, nfolds =
10, family = "multinomial")
pred.model.rrf10 = predict(model.rrf10, test.data, s = model.rrf10$lambda.min,
type="response")
cm.rrf10 = table(pred.model.rrf10, test.labels)
acc.rrf10 = sum(diag(cm.rrf10))/sum(cm.rrf10)
```

Macierz błędów dla nfolds = 10:

rrf10 \ test.labels	0	1	2	3	4	5	6	7	8	9
0	927	0	16	2	2	15	19	1	19	18
1	1	1083	32	8	6	16	13	18	18	8
2	10	1	838	41	12	7	26	26	34	11
3	2	3	22	736	12	26	1	15	16	22
4	2	0	12	2	805	24	18	13	20	49
5	9	2	1	117	14	697	42	4	44	16
6	10	4	50	12	20	15	827	1	16	0
7	8	0	19	32	4	25	1	895	13	77
8	10	41	32	45	22	47	10	1	738	5
9	1	1	10	15	85	20	1	54	56	803

Dokładność to 83,49%

Zweryfikujmy zatem, jak zmiana wartości parametru nfold wpłynie na uzyskaną dokładność.

Macierz błędów dla nfold = 3:

.rrf03	test.labels 0	1	2	3	4	5	6	7	8	9
0	930	0	16	4	3	17	23	1	21	18
1	1	1083	32	7	8	15	13	20	22	8
2	8	1	837	41	10	8	27	26	36	12
3	2	3	23	741	11	28	1	14	16	22
4	2	0	12	2	806	25	19	13	20	50
5	9	3	1	112	15	693	41	4	45	14
6	10	4	51	11	16	15	821	1	16	0
7	7	0	18	31	4	24	1	893	13	75
8	11	41	32	45	21	46	11	1	729	5
9	0	0	10	16	88	21	1	55	56	805

Dokładność to 83,38%

Macierz błędów dla nfolds = 30:

rrf30	test.labels 0	1	2	3	4	5	6	7	8	9
0	927	0	16	2	2	15	19	1	19	18
1	1	1083	32	8	6	16	13	18	18	8
2	10	1	838	41	12	7	26	26	34	11
3	2	3	22	736	12	26	1	15	16	22
4	2	0	12	2	805	24	18	13	20	49
5	9	2	1	117	14	697	42	4	44	16
6	10	4	50	12	20	15	827	1	16	0
7	8	0	19	32	4	25	1	895	13	77
8	10	41	32	45	22	47	10	1	738	5
9	1	1	10	15	85	20	1	54	56	803

Dokładność to 83,49%

Różnica uzyskanej dokładności jest minimalna, natomiast znacznie wzrósł czas obliczeń, stąd też wartość zostanie ustawiona na 3 podczas badania większego podzbioru MNIST.

Macierz błędów dla nfolds = 3 i podzbioru treningowego o wielkości 10 000 obrazów:

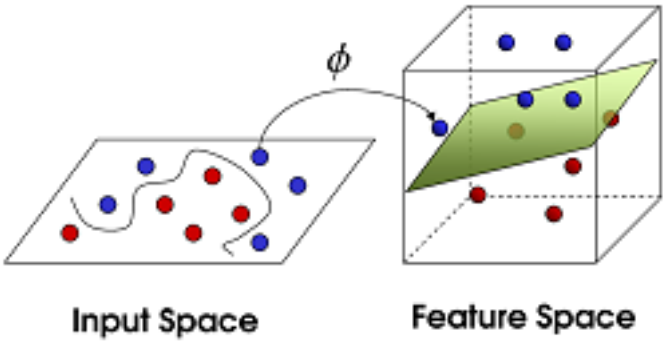
.rr	test.labels 0	1	2	3	4	5	6	7	8	9
0	946	1	7	4	1	8	8	2	5	8
1	1	1105	12	0	4	3	4	16	11	11
2	5	2	905	20	4	5	14	22	6	2
3	1	2	19	916	1	33	1	4	22	12
4	0	1	9	1	908	11	14	5	12	51
5	6	4	3	27	2	768	19	1	31	7
6	10	4	10	3	16	13	892	2	13	1
7	5	2	16	11	5	11	3	938	14	33
8	5	14	44	20	7	34	3	4	842	11
9	1	0	7	8	34	6	0	34	18	873

Dokładność to 90,93%

Mimo, że klasyfikatory opierające swoje działanie o regresje nie są zalecane dla problemów z wieloma klasami, regresja brzegowa spisuje się całkiem dobrze.
Zarówno dokładność, jak i charakterystyka macierzy błędów, są zbliżone do klasyfikatorów Random Forest i Gradient Boosted Trees.

7. Klasyfikator V – Support Vector Machine (SVM)

Maszyna wektorów nośnych to technika, która znajduje zastosowanie w wielu rodzajach problemów. Podobnie jak w przypadku regresji brzegowej, polega ona na wyznaczaniu hiperpłaszczyzny w wyższym wymiarze, która w jak najlepszym stopniu rozróżnia dwie klasy od siebie.



Aby móc zastosować ten klasyfikator dla problemu z wieloma klasami (w tym przypadku N = 10), musimy użyć jednej z dwóch technik:
- One vs Rest – jeden klasyfikator dla każdej z klas, czyli łącznie N klasyfikatorów. Ostateczną klasą zostaje ta, która uzyskała najwyższy wynik.
- One vs One – liczba klasyfikatorów wynosi N*(N-1)* ½, a ostateczną klasą zostaje ta, która dostała najwięcej głosów.

Jako że klasyfikator SVM jest wymagający obliczeniowo, w celu znalezienia optymalnego zestawu parametrów, użyto podzbioru o wielkości 5000 obrazów.

Kod odpowiedzialny za stworzenie modelu, klasyfikację i macierz błędów:

```
library(e1071)
model.svm1linear = svm(train.data[1:5000, ], train.labels[1:5000], kernel = "linear", cost
= 1, scale = FALSE)
pred.model.svm1linear = predict(model.svm1linear, test.data)
cm.svm1linear = table(pred.model.svm1linear, test.labels)
acc.svm1linear = sum(diag(cm.svm1linear))/sum(cm.svm1linear)
```

Macierz błędów SVM dla kernela 'linear':

	test.labels									
svm1linear	0	1	2	3	4	5	6	7	8	9
0	954	0	9	2	1	7	12	1	11	3
1	0	1118	10	2	1	2	2	15	7	9
2	2	1	921	17	5	3	17	27	20	3
3	1	2	18	894	0	28	1	6	39	5
4	2	1	20	3	937	13	16	13	16	60
5	7	3	8	49	0	796	20	3	40	12
6	6	3	9	2	5	11	885	0	8	0
7	2	0	11	12	4	2	0	915	9	31
8	3	6	24	14	1	22	3	4	801	5
9	3	1	2	15	28	8	2	44	23	881

Dokładność to 91,02%

Macierz błędów SVM dla kernela ‘radial’:

svm1radial	test.labels 0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	980	1135	1032	1010	982	892	958	1028	974	1009
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

Dokładność to 11,35%

Macierz błędów SVM dla kernela ‘polynomial’:

svm1polyno	test.labels 0	1	2	3	4	5	6	7	8	9
0	953	0	16	0	1	5	7	1	12	5
1	3	1113	18	9	6	11	7	20	1	10
2	3	2	951	11	2	2	3	17	5	1
3	0	2	9	938	0	17	1	3	21	9
4	0	0	4	1	944	2	9	8	13	30
5	8	0	2	20	0	829	9	0	16	4
6	9	6	8	1	5	10	920	1	3	0
7	2	0	13	12	5	2	0	953	8	8
8	2	12	8	10	0	9	2	0	886	4
9	0	0	3	8	19	5	0	25	9	938

Dokładność to 94,25%

Niezależnie od tego, w jaki sposób były zmieniane pozostałe parametry klasyfikatora, bądź same dane, dla kernela ‘radial’ klasyfikator za każdym razem typował tylko jedną klasę, stąd wynik bliski próbie losowej (10%).

Dla wartości ‘linear’ i ‘polynomial’, klasyfikator uzyskał sensowne wyniki. W przypadku ‘polynomial’, dokładność jest bliska klasyfikatorowi KNN i nie odbiegająca mocno od wyniku człowieka. Sprawdźmy, jak klasyfikator ten zachowa się w przypadku pełnego zbioru treningowego.

Macierz błędów SVM dla kernela ‘polynomial’ i pełnego zbioru treningowego:

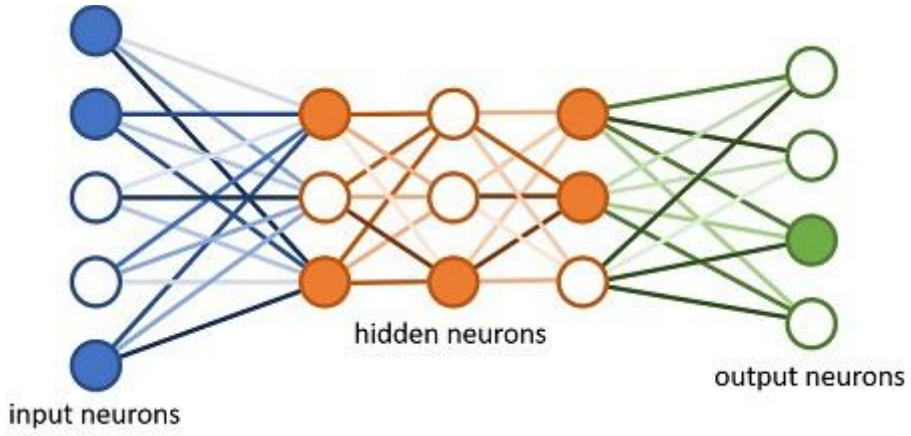
svm	test.labels 0	1	2	3	4	5	6	7	8	9
0	972	0	7	0	2	2	4	0	5	3
1	0	1124	0	2	0	0	5	9	0	5
2	1	2	1006	1	2	0	1	9	1	0
3	0	1	0	987	0	7	0	2	2	3
4	0	1	2	0	965	1	3	2	4	9
5	3	2	1	6	0	870	6	0	4	4
6	1	3	5	0	3	3	937	0	1	1
7	0	0	7	5	1	0	0	1000	4	1
8	2	2	3	6	0	4	2	0	950	3
9	1	0	1	3	9	5	0	6	3	980

Dokładność to 97,91%

SVM z takimi parametrami uzyskał wynik lepszy od człowieka i jest najdokładniejszy ze wszystkich badanych dotąd klasyfikatorów. Nie licząc cyfry 0 i 1, dystrybucja błędów w obrębie klas jest bardzo równomierna, co może sugerować, iż niewłaściwie sklasyfikowane obrazy byłby trudnie również dla człowieka.

8. Klasyfikator VI – Deep Neural Network

Ostatni z badanych klasyfikatorów to tzw. głęboka sieć neuronowa. Sieć neuronowa to struktura, która jest pewnym obrazem tego, w jaki sposób działa mózg. Sieć taka składa się z węzłów (neuronów) i połączeń między nimi. O tym, w jaki sposób dana wartość będzie propagowana do kolejnych węzłów, decydują wagi i funkcje aktywacji. Sieć może zawierać wiele ukrytych warstw.



Oto kod odpowiadający za stworzenie sieci z 1 ukrytą warstwą, wytrenowanie modelu i ewaluację:

```
library(mxnet)
dat = mx.symbol.Variable("data")
fc1 = mx.symbol.FullyConnected(dat, name = "fc1", num_hidden = 10)
smx = mx.symbol.SoftmaxOutput(fc1, name = "smx")

mx.set.seed(0)
model.nn1r10 = mx.model.FeedForward.create(
  smx,
  X = train.matrix.x,
  y = train.matrix.y,
  ctx = mx.cpu(),
  num.round = 10,
  array.batch.size = 100,
  learning.rate = 0.07,
  momentum = 0.9,
  eval.metric = mx.metric.accuracy,
  initializer = mx.init.uniform(0.07),
  epoch.end.callback = mx.callback.log.train.metric(100),
  array.layout = "rowmajor")

pred.model.nn1r10 = predict(model.nn1r10, test.matrix.x, array.layout = "rowmajor")
pred.model.nn1r10.label = max.col(t(pred.model.nn1r10))
cm.nn1r10 = table(pred.model.nn1r10.label, test.matrix.y)
acc.nn1r10 = sum(diag(cm.nn1r10))/sum(cm.nn1r10)
```

Macierz błędów DNN dla 1 ukrytej warstwy (64 neurony) i 10 iteracji:

	test.matrix.y									
nn1r10.label	0	1	2	3	4	5	6	7	8	9
1	957	0	4	3	1	8	11	1	9	10
2	0	1118	13	1	3	2	3	8	14	9
3	2	3	915	14	3	4	4	15	6	1
4	3	3	38	949	5	52	2	12	66	16
5	1	0	14	1	929	12	11	8	10	38
6	6	2	3	14	0	766	13	1	40	7
7	5	3	8	2	7	13	909	0	11	0
8	5	2	13	12	4	10	4	956	14	26
9	1	4	19	6	3	21	1	0	784	0
10	0	0	5	8	27	4	0	27	20	902

Dokładność to 91,85%

Uzyskana dokładność jest zadowalająca i bardzo zbliżona do inicjalnej w SVM. Można tu z pewnością wyróżnić cyfry, które sprawiały szczególne problemy.

Sprawdźmy zatem, jaki wynik osiągnie sieć, w której skład wchodzą 3 ukryte warstwy.

Macierz błędów DNN dla 3 ukrytych warstw (256, 128 i 64 neurony) i 10 iteracji:

	test.matrix.y									
nn4r10.label	0	1	2	3	4	5	6	7	8	9
1	972	0	2	0	1	4	7	0	6	3
2	0	1131	1	0	0	0	3	6	1	2
3	1	1	1017	14	1	1	1	14	7	1
4	0	0	3	980	0	11	1	1	7	7
5	0	0	1	0	965	1	4	0	4	14
6	0	0	0	2	0	862	5	0	5	2
7	2	0	1	0	5	5	936	0	1	0
8	4	2	4	10	1	2	0	1003	3	17
9	0	1	3	4	0	5	1	1	937	4
10	1	0	0	0	9	1	0	3	3	959

Dokładność to 97,62%.

Uzyskana dokładność jest identyczna z tą, jaką osiąga człowiek i nieznacznie gorsza od SVM przy najlepszej konfiguracji parametrów. Warto więc sprawdzić, jak zachowuje się ten klasyfikator, gdy zwiększona zostanie również ilość treningów (iteracji).

Macierz błędów DNN dla 3 ukrytych warstw (256, 128 i 64 neurony) i 25 iteracji:

	test.matrix.y									
nn4r25.label	0	1	2	3	4	5	6	7	8	9
1	965	0	1	1	0	2	1	2	3	2
2	0	1129	3	0	0	0	2	4	1	2
3	1	0	1012	2	3	0	0	6	5	0
4	1	1	3	992	1	6	0	2	6	4
5	0	0	1	0	948	1	2	0	0	3
6	2	1	1	2	0	879	18	0	3	2
7	8	1	2	0	5	1	934	0	1	1
8	0	2	5	6	4	0	0	1008	2	3
9	1	1	3	3	0	2	1	1	950	0
10	2	0	1	4	21	1	0	5	3	992

Dokładność to 98,02%

Jest to najlepszy wynik, jaki udało się dotąd uzyskać. Podczas obserwacji treningów, widać wyraźnie, że jest jeszcze trochę miejsca na poprawę.

Macierz błędów DNN dla 3 ukrytych warstw (256, 128 i 64 neurony) i 50 iteracji:

	test.matrix.y									
nn4r50.label	0	1	2	3	4	5	6	7	8	9
1	971	0	2	0	0	2	3	1	2	1
2	1	1130	0	1	0	0	2	1	0	2
3	1	1	1015	3	2	0	1	8	5	0
4	1	0	2	992	0	5	1	1	2	5
5	0	0	2	0	963	1	2	0	1	6
6	1	1	0	4	0	880	3	0	1	2
7	2	0	1	0	4	1	944	0	2	1
8	0	1	4	5	3	0	0	1014	2	2
9	1	2	6	3	1	2	2	0	956	1
10	2	0	0	2	9	1	0	3	3	989

Dokładność to 98,54%

Wynik został poprawiony o pół punkta procentowego, co w odniesieniu do poprzedniej dokładności (98%) oznacza, że sieć ta popełnia aż o 25% mniej błędów.
Przeciętny człowiek, stając przed zadaniem klasyfikacji zbioru MNIST, popełni 60% więcej błędów.

Po około 30 iteracjach, sieć uzyskała 100% poprawności na danych treningowych, powodując, że kolejne iteracje były zbędne. Celem dalszego poprawienia dokładności, koniecznie byłoby dostarczenie większej ilości danych treningowych.

9. Podsumowanie

Jak otrzymany wynik ma się do istniejących rekordów dla zbioru MNIST?
W dniu pisania tego sprawozdania, najlepszą dokładnością, jaką odnotowano, jest **99,79%** poprawnie sklasyfikowanych obrazów. Rekord ten został ustanowiony przez konwolucyjną sieć neuronową o 6 warstwach. Wysokie wyniki uzyskały również algorytmy K-najbliższych sąsiadów (99,48%) oraz SVM (99,44%).

Źródła

1. Materiały z laboratoriów
2. Dokumentacja R
3. Kanał StatQuest na portalu YouTube
4. Opracowania na portalu Rpubs
5. Zbiór MNIST w CSV:
https://www.kaggle.com/oddrationale/mnist-in-csv

Załączniki

1. Kod źródłowy w R (script.R)
2. Grafiki i pliki tekstowe z macierzami błędów