Rozpoznawanie cyfr

Wstęp

Niniejszy projekt dotyczy rozpoznawania cyfr w postaci odręcznego zapisu. Za pomocą algorytmów genetycznych postaramy się zbadać, czy i w jakim stopniu możliwe jest odczytanie takich znaków. Zbadamy także który z przetestowanych algorytmów jest najlepszy, porównamy jego osiągi z pozostałymi wykorzystanymi metodami, a także spróbujemy swoich sił z najlepszymi rozwiązaniami powyższego problemu umieszczonymi na stronie internetowej Kaggle.

Narzędzia

Zdecydowałem się na wykorzystanie języka R jako podstawowego narzędzia do wykonywania poszczególnych operacji na danych. Pakiet o nazwie RStudio pozwala na wygodne i efektywne wykorzystanie możliwości języka, a także umożliwia szybki dostęp do internetowej bazy wiedzy na temat wszystkich dostępnych metod.
Wszelkie wykorzystane algorytmy są częścią ogólnodostępnych, oficjalnych bibliotek dostępnych w sieci CRAN (Comprehensive R Archive Network).
Aby uruchomić załączony skrypt należy zainstalować następujące pakiety:
- e1071 - zawiera metody wykorzystujące Support Vector Machines,
- randomForest - biblioteka Random Forest,
- readr - pomocny w odczycie/zapisie danych,
- scales - zawiera m.in. procent,
- party - jego częścią jest algorytm Conditional Inference Trees,
- class - pod tym hasłem kryje się m.in. algorytm k Nearest Neighbours.

za pomocą komendy `install.packages("$nazwa_pakietu")`, a następnie załadować je używając `library($nazwa_pakietu)`.

Dane

Projekt został wykonany przy wykorzystaniu świetnie przygotowanej bazy danych dostępnej na stronie Kaggle. Udostępniony przez ekipę Kaggle plik `test.csv` zawiera 42000 zakodowanych obrazów, a każdy z nich przedstawia ręcznie pisaną cyfrę (białe znaki na czarnym tle). Wszystkie liczby reprezentowane są przez ciąg pikseli (ponumerowanych od 0 do 783), które to są odpowiednio oznaczone wartością od 0 do 255 odpowiadającą kolorowi danego piksela w skali szarości (0 to czerń, 255 oznacza biel). Opierając się na tych informacjach możemy stwierdzić, że na przykład piksel o numerze 522 z wartością 255 odpowiada jednemu białemu punktowi na obrazku, czyli zapewne stanowi część ręcznie zapisanej cyfry.

Będziemy operować na pliku testowym, a więc każdy wiersz zawiera swego rodzaju rozwiązanie zagadki, czyli wartość logiczną zakodowanej cyfry. Rozwiązanie będzie oczywiście "zatajone" przed naszym programem w momencie rozpoznawania poszczególnych wartości z właściwego zbioru testowego.
Podsumowując - mamy do czynienia z naprawdę dobrze przygotowaną bazą. Dane są w przemyślany sposób przedstawione i znormalizowane. Ilość dostępnych rekordów pozwala na pełną precyzję pomiarów i pozwoli na należyte przeprowadzenie pracy badawczej.

Należy zauważyć jednak, że plik z danymi mający 42000 rekordów składających się z niemal 800 kolumn stanowi ogromne wyzwanie nawet dla nowoczesnych komputerów. Pojawia się więc konieczność swego rodzaju "skompresowania" danych, aby nasze algorytmy klasyfikujące były w stanie przeprocesować rekordy testowe w rozsądnym czasie.
Mamy dwie możliwości - możemy "skrócić" naszą bazę w pionie (liczba rekordów), bądź też w poziomie (redukcja ilości kolumn z danymi).
Pierwsza droga polegająca na ograniczeniu ilości rekordów powoduje drastyczne spadki w precyzji naszych obliczeń. Algorytmy klasyfikujące nie miałyby dostatecznie dużych zbiorów testowych, co jest głównym czynnikiem wpływającym na dokładność pomiaru. Dlatego też zdecydowałem się na "skompresowanie" danych poprzez zredukowanie liczby kolumn w rekordach poprzez zastosowanie algorytmu PCA (ang. Principal Component Analysis, a więc Analiza Głównych Składowych).
W ten sposób zamiast korzystać z 784 kolumn z pikselami mogłem oprzeć się na 25 głównych składowych tychże danych.
Swoista macierz tworzona przez wymiary pliku testowego została więc zredukowana z 42000x785 na 42000x26 co wygląda znacznie lepiej z wydajnościowego punktu widzenia.
Tak przygotowane dane są doskonałym wyjściem do eksperymentów.

Opis eksperymentów

Wszystkie pliki źródłowe, ilustrujące krok po kroku, są dostępne w paczce z niniejszym sprawozdaniem. Mamy do dyspozycji kilka skryptów:

- ClassifyUsingPca.r - plik zawierający wszystkie testy klasyfikatorów na głównych składowych bazy danych,
- RandomForestRawData.r - skrypt uruchamiający algorytm Random Forest na oryginalnym zestawie danych,
- SvmTestForKaggle.r - zestaw poleceń, które posłużyły mi do wzięcia udziału w konkursie na stronie Kaggle, algorytm Support Vector Machines uruchomiony na 50 głównych składowych danych konkursowych.

Wszystkie pomiary zostały przeprowadzone na wcześniej wymienionej bazie danych zaczerpniętej z portalu Kaggle. Dla przypomnienia - mamy do dyspozycji 42000 rekordów, które zostały potraktowane algorytmem PCA i ograniczone do 25 głównych składowych. Zdecydowałem się na podział zbioru na treningowy i testowy według proporcji 0,738: 0,262. Oznacza to, że każdy algorytm klasyfikujący miał do dyspozycji zbiór treningowy w postaci 31000 rekordów wraz z rozwiązaniami. Po "nauczeniu się'

danych klasyfikator przystępował do właściwych testów - operując na pozostałych 11000 obiektach starał się im dopasować kształty poszczególnych cyfr. Wyniki pracy algorytmów następnie zostały zestawione z rzeczywistymi rozwiązaniami pobranymi z oryginalnej bazy danych. Otrzymaliśmy w ten sposób wartość precyzji każdego algorytmu - procent właściwie rozpoznanych cyfr, a także macierz błędu pozwalającą na dokładnie przeanalizowanie wyników. Dzięki temu będziemy mogli stwierdzić, że na przykład dany klasyfikator ciągle mylił cyfrę '3' z kształtem "ósemki", a niejako w zamian nie miał najmniejszego problemu z rozpoznaniem "jedynki".
Aby mieć pełny pogląd na sytuację zdecydowałem się również spróbować uruchomić poszczególne klasyfikatory na nieskompresowanym, 785-kolumnowym zbiorze danych. Czy któryś z nich podołał takiemu obciążeniu? Sprawdzimy to w dalszej części naszego eksperymentu.

Wyniki

Zacznijmy od przedstawienia wyników klasyfikacji na bazie 25 głównych składowych. Dla przypomnienia - każdy algorytm "uczy się" na 31000 rekordów, a następnie jest testowany na pozostałych 11000. Spójrzmy na rezultaty:

Random Forest

Precyzja: 93,7%
Macierz błędu:

Actual\Predicted	0	1	2	3	4	5	6	7	8	9
0	1058	0	1	1	1	0	9	0	4	2
1	0	1194	6	4	2	0	5	1	4	2
2	8	3	1102	5	6	1	3	13	2	3
3	1	1	12	1033	0	34	6	6	9	5
4	1	3	4	1	1015	0	8	1	2	33
5	7	3	3	13	0	958	8	1	14	5
6	5	1	2	0	4	12	1086	1	2	0
7	1	6	12	3	8	2	0	1080	4	21
8	3	4	9	19	4	14	4	0	965	22
9	8	5	3	8	20	5	1	12	12	1005

5 Nearest Neighbours

Precyzja: 97,1%

Macierz błędu:

Actual\Predicted	0	1	2	3	4	5	6	7	8	9
0	1066	0	0	0	0	0	8	1	1	0
1	0	1205	5	2	1	0	1	0	1	3
2	5	3	1115	2	1	0	2	12	6	0
3	0	2	9	1063	0	12	1	2	13	5
4	0	4	0	0	1025	1	4	3	1	30
5	1	1	0	13	1	975	12	3	2	4
6	4	0	0	0	2	3	1104	0	0	0
7	0	8	7	0	3	1	0	1101	1	16
8	1	2	1	13	0	13	5	3	998	8
9	1	1	1	7	17	2	3	16	4	1027

Conditional Inference Trees

Precyzja: 82,7%

Macierz błędu:

Actual\Predicted	0	1	2	3	4	5	6	7	8	9
0	963	0	15	18	2	36	18	9	11	4
1	0	1160	16	7	1	15	4	5	9	1
2	18	10	936	54	20	15	29	26	24	14
3	16	5	49	905	8	58	5	9	43	9
4	6	6	22	4	850	20	17	19	15	109
5	32	11	15	103	14	758	19	11	33	16
6	11	7	43	9	18	22	975	6	9	13
7	7	8	26	8	24	9	4	973	15	63
8	13	13	44	87	17	70	10	11	763	16
9	5	4	6	17	97	36	8	68	19	819

Naive Bayes

Precyzja: 85,6%

Macierz błędu:

Actual\Predicted	0	1	2	3	4	5	6	7	8	9
0	991	0	7	4	3	33	26	2	3	7
1	0	1154	27	8	1	14	6	2	5	1
2	8	13	956	29	16	16	39	23	30	16
3	5	8	25	893	0	92	8	9	43	24
4	0	12	6	0	922	15	9	2	9	93
5	6	4	13	74	15	835	21	11	25	8
6	10	3	7	0	5	86	995	0	6	1
7	3	31	31	4	30	5	2	950	14	67
8	7	11	18	64	5	54	8	7	842	28
9	6	9	14	13	69	26	7	34	18	883

Support Vector Machines

Precyzja: 97,5%

Macierz błędu:

Actual\Predicted	0	1	2	3	4	5	6	7	8	9
0	1063	0	2	0	0	1	9	0	1	0
1	0	1203	5	3	2	0	0	1	2	2
2	2	4	1118	3	3	0	0	9	5	2
3	0	0	9	1073	0	12	3	3	5	2
4	0	1	0	0	1039	0	4	0	1	23
5	1	1	0	9	1	986	8	1	4	1
6	3	0	0	0	3	3	1100	0	4	0
7	1	5	8	2	4	0	1	1100	3	13
8	0	2	5	7	0	9	4	2	1011	4
9	2	3	1	2	13	5	2	11	3	1037

Na poprzednich stronach mamy przedstawione wyniki dla zbioru danych testowych potraktowanych algorytmem PCA, a więc właściwie skompresowanego i dość "lekkiego" w obliczeniach dla współczesnych komputerów.
Co stanie się jednak, jeśli spróbujemy uruchomić nasze klasyfikatory na oryginalnej, 785-kolumnowej bazie 42000 rekordów?
Okazuje się, że tak wielkiemu zbiorowi danych jest w stanie podołać w sensownym czasie jedynie algorytm Random Forest. Próby klasyfikowania nieskompresowanej bazy z wykorzystaniem pozostałych testowanych sposobów niestety się nie powiodły - nie udało mi się doczekać wyników.

Przedstawmy jedyne kompletne wyniki bez użycia PCA:

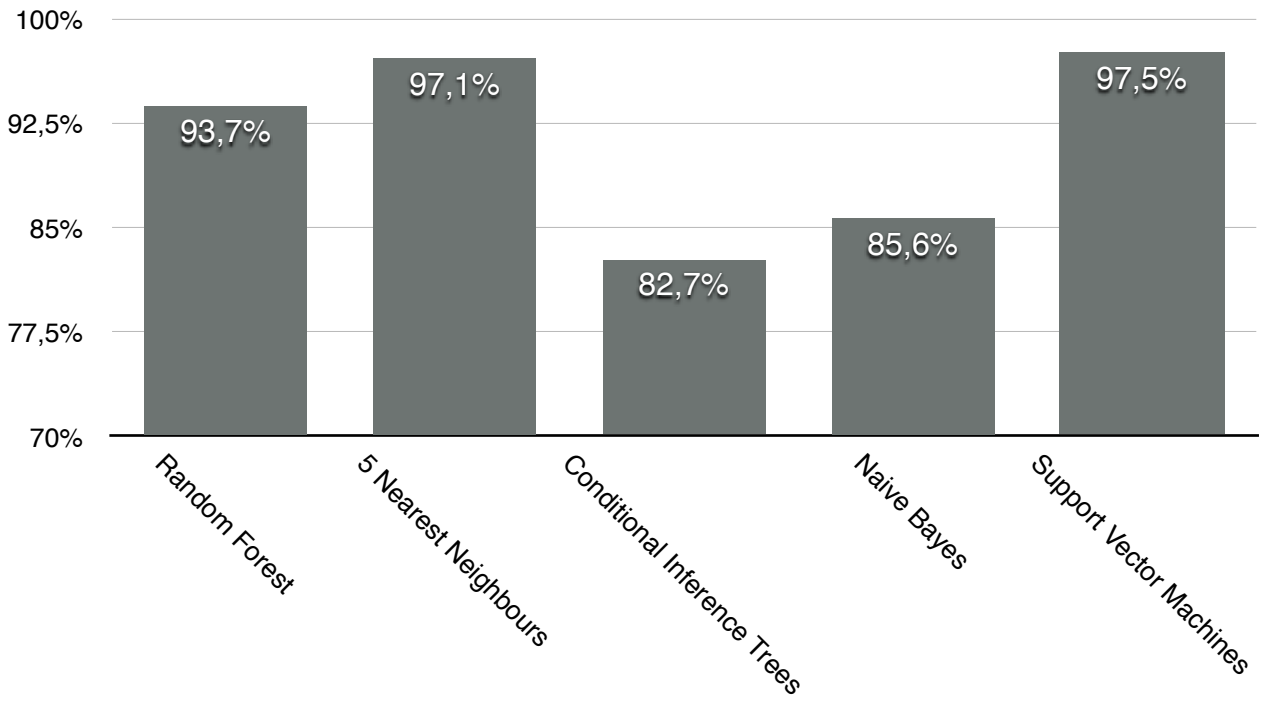
Random Forest na oryginalnej bazie danych

Precyzja: 95,4%
Macierz błędu:

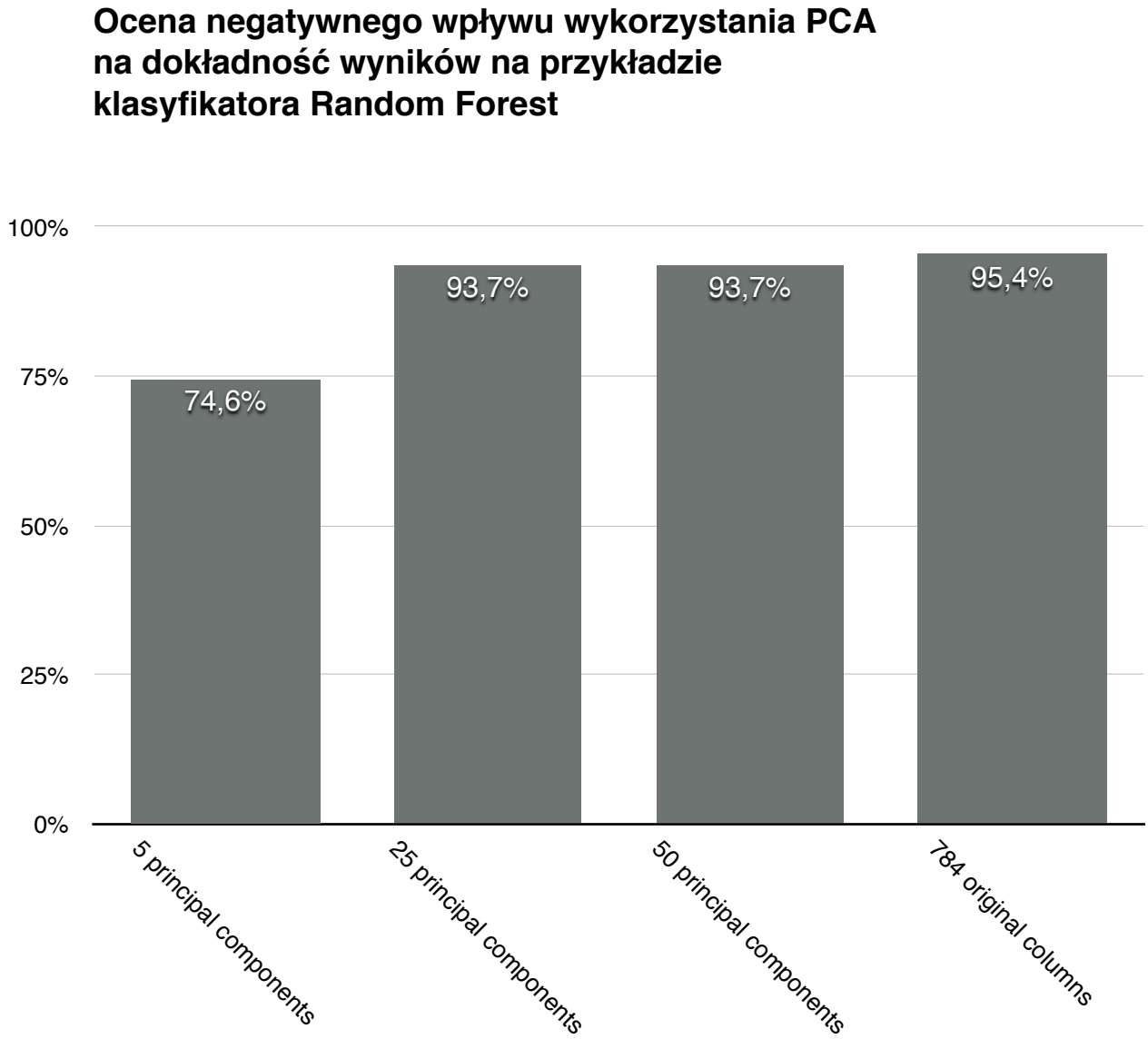
Actual\Predicted	0	1	2	3	4	5	6	7	8	9
0	1040	0	1	6	2	1	17	1	5	3
1	0	1190	5	3	1	4	5	1	7	2
2	5	6	1060	14	7	5	5	16	20	8
3	3	1	17	1009	1	21	5	4	32	14
4	0	5	3	3	997	2	9	3	6	40
5	3	1	6	17	3	952	9	2	13	6
6	9	1	2	0	3	13	1078	1	6	0
7	0	6	24	3	5	0	1	1067	3	28
8	3	4	8	35	3	29	3	5	940	14
9	3	4	1	13	36	8	3	27	11	973

Interpretacja wyników

Jak można wywnioskować z powyższych wyników właściwe przygotowanie danych za pomocą PCA potrafi zdziałać cuda. Nawet prosty klasyfikator K Najbliższych Sąsiadów osiągnął wynik na poziomie 97%.
Można zadać pytanie, jak radziły sobie klasyfikatory na mniejszej ilości danych, ale w oryginalnych 785 kolumnach? Otóż przy nauce na 3100 rekordach i testach na 1100 Random Forest, kNN i SVM osiągnęły precyzję na poziomie ~70%.
Conditional Inference Trees i Naive Bayes wypluwały już dużo gorsze wyniki, w okolicach 30-50%. Innymi słowy stanowczo nie powinniśmy iść w tę stronę.
Jedyne wyjście to "skompresowanie" liczby kolumn, w czym pomoże algorytm PCA.
Poniżej porównanie precyzji wszystkich klasyfikatorów:



Mając na uwadze fakt, że algorytm Random Forest poradził sobie jako jedyny w rozsądnym czasie z klasyfikacją całego zestawu danych w oryginalnej postaci możemy również ocenić negatywny wpływ PCA na wyniki. Jak widać na poniższym wykresie już przy tylko 5 głównych składowych osiągana precyzja nie jest tragiczna, a wybierając 25 i więcej mamy dokładność niemalże taką, jak na oryginalnej bazie. Nie muszę chyba przypominać niesamowitego wzrostu wydajności operacji na w ten sposób skompresowanych danych. Działając na 25 kolumnach zamiast 784 oryginalnych, realnie działamy na ~30 razy "węższym" zbiorze rekordów, a mimo to osiągamy wyniki na poziomie 97%.



Warto jeszcze przeanalizować macierze błędu każdego z algorytmów. Po uśrednieniu tych wartości możemy znaleźć kilka interesujących faktów. Dla naszych klasifykatorów zdecydowanie najłatwiejsza w rozpoznaniu okazała się cyfra '1'. Nawet najmniej precyzyjny pomiar oparty na Conditional Inference Trees osiągnął precyzję dla cyfry '1' na poziomie 95,2%.
Które z cyfr okazały się najtrudniejsze do rozpoznania? Mamy widocznych kilka powtarzających się pomyłek. Najczęściej uzyskane problemy z precyzją to:
- '9' błędnie oznaczane jako '4',
- '5' oznaczane jako '3',
- '8' jako '3',
- '8' jako '5',
- '7' jako '9'.

Udział w konkursie Kaggle

Skoro uzyskaliśmy przy pomocy klasyfikatora Support Vector Machines precyzję na poziomie 97,5% przy wykorzystaniu 3/4 danych testowych do nauki, dodatkowo jeszcze potraktowanych PCA to zdecydowanie warto byłoby się pokusić o wzięcie udziału w konkursie "Digit Recognizer" na stronie Kaggle.
Na potrzeby wygenerowania w miarę możliwości najlepszego wyniku zdecydowałem się na uruchomienie algorytmu SVM na zbiorze danych ograniczonym do 50 głównych składowych. Dalsze powiększanie liczby głównych składowych skutkowało już efektem odwrotnym od zamierzonego, czyli pogorszeniem wyników
Przyuczenie klasyfikatora na pełnym zbiorze 42000 rekordów treningowych, a następnie wykonanie pomiarów na oficjalnych 28000 wierszach testowych dało ogólną precyzję na poziomie 98,257%. Uzyskany rezultat daje mi - w chwili sporządzania niniejszego tekstu - 248. miejsce z 1000 osób na tablicy wyników.