Laboratorium Rozpoznawania Obrazów – Ćwiczenie #3 & #4
Rozpoznawanie cyfr pisanych ręcznie

Termin oddawania: **15.04, 22.04**

W tym ćwiczeniu Państwa zadaniem będzie rozpoznawanie cyfr pisanych ręcznie za pomocą zestawu klasyfikatorów liniowych. W zadaniu wykorzystamy dane z bazy danych MNIST (http://yann.lecun.com/exdb/mnist). Zbiory uczący i testowy można pobrać spod podanego wyżej adresu lub z serwera Galera (http://galeranew.ii.pw.edu.pl/~rkz/rob/mnist.zip).

Warto wiedzieć, że obrazy cyfr wydzielone z zeskanowanej strony są normalizowane w następujący sposób:

1. Prostokąt zawierający czarno-biały obraz znaku zeskanowanego w rozdzielczości 300 dpi jest skalowany proporcjonalnie do prostokąta o większym z wymiarów równym 20. W trakcie skalowania obraz jest zamieniany na skalę szarości (proporcjonalnie do liczby oryginalnych pikseli pierwszego planu przypadających na jeden piksel obrazu po przeskalowaniu).
2. Jest wyznaczany środek ciężkości przeskalowanego znaku, a znak jest umieszczany w obrazie 28x28 pikseli tak, żeby środek ciężkości znalazł się na środku tego większego obrazu.

Państwa zadaniem, jest przygotowanie klasyfikatora korzystającego z **klasyfikatorów liniowych** rozróżniających konkretne cyfry. Oprócz jakości klasyfikacji na zbiorze testowym należy podać (dla wybranych przypadków) **macierz pomyłek**.

Dodatkowe zadanie (warte 4 punkty z 12) polega na poprawieniu jakości klasyfikacji w stosunku do klasyfikacji zrealizowanej przez zwykłe głosowanie klasyfikatorów elementarnych. Nie chodzi przy tym o prostą zmianę głosowania (np. zamiast jednomyślnego – większość absolutna): możliwe rozwiązania proszę rozważyć po przeanalizowaniu wyników eksperymentów z pierwszej części ćwiczenia.

Rozwiązaniem referencyjnym, jest klasyfikator głosujący (wykorzystujący głosy 45 klasyfikatorów liniowych *one vs. one*) wskazujący jako wynik konkretną cyfrę, o ile zostało oddanych na nią 9 głosów. W pozostałych przypadkach jest podejmowana decyzja wymijająca. Wyniki klasyfikacji podsumowuje poniższa tabela (choć ciekawy wgląd w klasyfikację może dać analiza macierzy pomyłek); klasyfikatory działały na 40 składowych głównych.

	Zbiór uczący MNIST			Zbiór testowy MNIST		
	OK.	Błąd	Odrzucenie	OK.	Błąd	Odrzucenie
Jakość klasyfikacji	91.34%	5.72%	2.94%	91.55%	5.49%	2.96%

Zadanie można podzielić na kilka części:

1. Przygotowanie podstawowego algorytmu wyznaczającego parametry płaszczyzny decyzyjnej dla zadanego zbioru uczącego, zawierającego dwie klasy. W zupełności wystarczy użycie tutaj algorytmu uczenia perceptronu, ale możecie Państwo poeksperymentować i z innymi metodami wyznaczania płaszczyzny decyzyjnej.
2. Testy algorytmu dla wielowymiarowych danych cyfr. Przeprowadzenie tego testu ma na celu sprawdzenie wydajności algorytmu oraz jego zachowania w przestrzeni

wielowymiarowej. Ze względu na kolejne punkty wymiarowość trzeba zredukować do 40-60 składowych używając PCA.

3. Rozwiązanie podstawowe polega na przygotowaniu 45 klasyfikatorów – po jednym dla każdej pary cyfr – i przeprowadzeniu głosowania klasyfikatorów na poszczególne cyfry. Dostarczony kod przeprowadza głosowanie jednomyślne (tzn. wszystkie 9 klasyfikatorów „znających" konkretną cyfrę musi być zgodnych co do klasy). Możecie Państwo zmienić tę zasadę, ale w kolejnych eksperymentach trzymajcie się takiej samej metody głosowania. W sprawozdaniu proszę umieścić jakość klasyfikacji pojedynczych klasyfikatorów oraz wyniki klasyfikacji cyfr.
4. Przygotowanie zespołu klasyfikatorów *one-versus-rest*. Trzeba pomyśleć o uczeniu zespołu (funkcja podobna do `trainOVOensemble`), a wcześniej o reprezentacji zespołu. Przy reprezentacji macierzowej (jak w OVO, np. z ujemną drugą etykietą) można będzie wykorzystać z minimalnymi zmianami funkcje `voting` i `unamvoting`. Alternatywą jest oczywiście napisanie całego osobnego toru uczenie-głosowanie-klasyfikacja dla zespołu OVR.
5. Sprawdzenie jakości klasyfikacji OVR na całym zbiorze uczącym. Proszę podać jakość pojedynczych klasyfikatorów oraz całego zespołu.
6. Zwiększenie liczby cech funkcją `expandFeatures`. Dodaje ona cechy, które są iloczynami oryginalnych cech: $x_i x_j \; for \; i \leq j$. Proszę zwrócić uwagę, że jeśli F to początkowa liczba cech, to po "ekspansji" nowych cech będzie $F + \frac{F(F-1)}{2}$. Przy 40 składowych PCA dostaniemy 40 + 780 nowych cech (czyli w sumie 860).
7. Powtórka punktów 3-6 dla rozszerzonego zestawu cech.
8. Ostatni krok, to usprawnienie najlepszego z dotychczasowych rozwiązań. Jedynym ograniczeniem jest używanie klasyfikatorów liniowych w przestrzeni cech.

W sprawozdaniu proszę zamieścić:

1. Krótki opis metody klasyfikacji znaków przy użyciu klasyfikatorów liniowych.
2. Krótki algorytmu wyznaczania parametrów płaszczyzny decyzyjnej.
3. Dane dotyczące jakości klasyfikacji każdego z wykorzystywanych klasyfikatorów liniowych.
4. Dane dotyczące jakości klasyfikacji cyfr, z wnioskami wynikającymi z analizy **macierzy pomyłek** (proszę zachować umiar w umieszczaniu macierzy pomyłek w sprawozdaniu).
5. Opis usprawnienia wprowadzonego do najlepszego rozwiązania „standardowego"; porównanie wyników obu wersji. Byłoby ideałem, gdyby dodać parę zdań uzasadnienia dlaczego uważacie, że faktycznie jest to usprawnienie.

Do pakietu sprawozdanie + kod proszę **nie dołączać danych** (koszt dołączenia danych wynosi 2 punkty)!

Osoby zainteresowane mogą sprawdzić, jakie wyniki klasyfikacji osiągną na zbiorze danych przygotowanym ze znaków pisanych lokalnie (tzn. w Warszawie). Zestaw danych w formacie MNIST jest dostępny pod adresem: http://galeranew.ii.pw.edu.pl/~rkz/rob/pldigits.zip. O skutkach użycia zbioru uczącego pochodzącego z innej populacji, niż dane faktycznie klasyfikowane mówiłem, ale warto zobaczyć skutki na własne oczy.

Lista skryptów w zestawie:

`compErrors.m`	- liczy współczynniki rozpoznawania na podstawie macierzy pomyłek
`confMx.m`	- tworzy macierz pomyłek na podstawie wektorów etykiet faktycznych i wyniku klasyfikacji
`expandFeatures.m`	- liczy nowy zestaw cech mnożąc przez siebie cechy wejściowe
`mainscript.m`	- skrypt główny
`pcaTransform.m`	- liniowa transformacja cech do nowej przestrzeni
`perceptron.m`	- wyznaczenie płaszczyzny separującej klasy algorytmem perceptronu
`prepTransform.m`	- wyznaczenie parametrów transformacji PCA dla zadanej liczby składowych głównych
`readmnist.m`	- odczyt pliku obrazów i pliku etykiet w formacie MNIST
`readSets.m`	- odczyt zbioru uczącego i testowego bazy MNIST (korzysta z readmnist)
`trainOVOensamble.m`	- uczenie zestawu klasyfikatorów dla zespołu one vs. one
`trainSelect.m`	- uruchamia kilka razy perceptron i wybiera najlepsze (w sensie najmniejszej liczby błędów) rozwiązanie
`unamvoting.m`	- jednomyślna decyzja klasyfikacyjna na podstawie macierzy głosów klasyfikatorów (woła voting)
`voting.m`	- wypełnienie tablicy (macierzy) głosów klasyfikatorów w zespole