Metody systemowe i decyzyjne w informatyce

Laboratorium – ZALICZENIE – Zadanie nr 4

Rozpoznawanie ręcznie pisanych znaków

autorzy: A. Gonczarek, P. Klukowski, J. Tomczak, S. Zaręba, M. Zięba, J. Kaczmar

Cel zadania

Celem zadania jest implementacja modelu, który pozwoli na klasyfikację miniatur zdjęć reprezentujących ubrania (przykładowe obrazy przedstawiono na rysunku 1). Aby skomplikować zadanie, dane zostały lekko zniekształcone poprzez dodanie szumu i wykonanie drobnych przesunięć obiektów, które znajdują się na zdjęciach. Wybór modelu jest dowolny, podobnie jak algorytmu uczenia, oraz ekstrakcji cech. Zaimplementowane podejście oceniane będzie poprzez poprawność predykcji (klasyfikacji) na próbie testowej.

W ramach zadania do wykonania są następujące kroki:

- Ekstrakcja cech z obrazu.
- Wybór i implementacja modelu.
- Wybór i implementacja algorytmu uczenia.
- Predykcja dla nowego obrazu w oparciu o wybrany model.

Rysunek 1: Przykładowe miniatury zdjęć reprezentujących ubrania (przed dodaniem szumu).

Uwaga! Nie wszystkie z powyższych etapów muszą być koniecznie wykonane. Przykładem jest klasyfikator κ-NN, który może działać bezpośrednio na obrazach bez ekstrakcji cech oraz nie wymaga stosowania algorytmu uczenia, gdyż do predykcji wykorzystuje bezpośrednio ciąg uczący.

W kolejnych podpunktach wskazane zostaną **przykładowe** rozwiązania poszczególnych etapów. Nie oznacza to, że należy ograniczać się wyłącznie do nich!

Ekstrakcja cech

Dla skutecznej klasyfikacji zazwyczaj stosuje się **ekstrakcję cech** (ang. *feature extraction*) z obiektu (np. obrazu). Ekstrakcja cech polega na przetworzeniu obiektu do zestawu wielkości (cech), które reprezentują istotne informacje o obiekcie. Istnieje wiele sposobów ekstrakcji cech, natomiast dobór odpowiedniej metody jest zależny od rozpatrywanego zagadnienia, np. ekstrakcja cech z obrazów, sygnałów dźwiękowych, sygnałów medycznych. W dalszej części przedstawiono przykładowe metody ekstrakcji cech.

Filtry Gabora. Jedną z metod ekstrakcji cech z obrazu są *filtry Gabora*. Idea filtrów Gabora opiera się na określoniu pewnych funkcji bazowych złożonych z funkcji gaussowskiej i sinusoidalnych. Wówczas, nakładając filtry Gabora na obraz dostajemy odpowiedz, która przekazywana jest do klasyfikatora. Więcej o filtrach Gabora można przeczytać np. w pracy: `http://personal.lut.fi/users/joni.kamarainen/downloads/publications/ipta2012.pdf`

Filtry Haara. Idea *filtrów Haara* polega na wykorzystaniu prostych obrazów, zazwyczaj prostokątnych, w których określa się jasne i czarne obszary, a następnie liczy się różnice między sumą pikseli w odpowiednich obszarach. Filtry te zawdzięczają swoją popularność dzięki łatwości implementacji oraz szybkości wyznaczenia cech przy zastosowaniu tzw. *obrazów całkowych (ang. integral images)* . Więcej o filtrach Haara można przeczytać np. na stronie: `http://en.wikipedia.org/wiki/Haar-like_features`, oraz w pracy: `http://www.merl.com/reports/docs/TR2004-043.pdf`

Funkcje jądra. Odmiennym sposobem określania cech od filtrów Gabora czy Haara jest wykorzystanie zaobserwowanych obiektów. Przykładem takiego podejścia jest wybranie pewnego podzbioru M zaobserwowanych obrazów (sposób wyboru może być losowy lub metodyczny) oraz zdefiniowanie funkcji określającej podobieństwo między nowym obiektem a znanymi obiektami $k(\mathbf{x}^{new}, \mathbf{x}_n)$. Funkcja ta, jeśli spełnia pewne określone własności, nazywana się **funkcją jądra** (ang. *kernel function*, lub *kernel*). Wówczas wartość funkcji jądra dla każdego zapamiętanego obiektu jest cechą, a w rezultacie otrzymujemy ciąg M cech. Podobna idea wykorzystywana jest w tzw. radialnych sieciach neuronowych: `http://en.wikipedia.org/wiki/Radial_basis_function_network`

Analiza składowych głównych. Do ekstrakcji cech z obrazu można również wykorzystać analizę składowych głównych (ang. *Principal Component Analysis*, PCA). Idea PCA polega na znalezieniu przekształcenia liniowego wysokowymiarowej (oryginalnej) przestrzeni na niskowymiarową przestrzeń cech. Opis zastosowania PCA do ekstrakcji cech z obrazów można znaleźć np. w prezentacji `http://cmp.felk.cvut.cz/~hlavac/TeachPresEn/11ImageProc/15PCA.pdf`

Ekstrakcja ręcznie określonych cech. Czasami znajomość problemu pozwala na ręczne określenie cech. Przykładowo, dla ręcznie pisanych liter lub cyfr można zaproponować następujące cechy:

- Stosunek wysokości fragmentu symbolu do wysokości całego symbolu.
- Stosunek szerokości fragmentu symbolu do szerokości całego symbolu.
- Gęstość pikseli w wybranych fragmencie obrazu.

Niemożliwe jest określenie ogólnej metodyki proponowania cech, ponieważ jest ona zależna od zagadnienia oraz pomysłowości projektanta.

Warto zaznaczyć, że różne techniki ekstrakcji cech można ze sobą łączyć tworząc bogatszy zbiór cech, a także można je łączyć w sposób potokowy, np. wektor cech uzyskany przy pomocy filtrów Haara przekształcić z użyciem PCA.

Model

Modele generujące. Korzystanie z *modeli generujących* (ang. *generative modeling*) ma na celu modelowanie rozkładu łącznego zmiennych losowych $\mathbf{x}$ (cechy) i y (klasa) za pomocą $p(\mathbf{x}, y|\boldsymbol{\theta})$, gdzie $\boldsymbol{\theta} \in \mathbb{R}^M$ oznacza wektor parametrów modelu. Zauważmy, że modele generujące pozwalają na faktoryzację rozkładu łącznego:[1]

$$p(\mathbf{x}, y) = p(\mathbf{x}|y)p(y) \tag{1}$$

$$= p(y|\mathbf{x})p(\mathbf{x}). \tag{2}$$

Znając rozkład łączny cech i wyjść można najpierw wygenerować wartość klasy, a następnie wartości cech (wzór (1)), lub w pierwszej kolejności wygenerować wartości cech, a później wartość klasy (wzór (2)). Dlatego też modele dla rozkładu łącznego nazywane są *generujące*, ponieważ umożliwiają generowanie zarówno wartości klas, jak również cech.

Przykładami modeli generujących są:

- *Gaussian Discriminant Analysis* (GDA): gdy cechy w poszczególnych klasach modelowane są rozkładami normalnymi $p(\mathbf{x}|y)$, a rozkład na klasy $p(y)$ modelowany jest rozkładem wielopunktowym;
- *Naive Bayes*: podobnie jak GDA, z dodatkowym założeniem, że cechy są niezależne (diagonalne macierze kowariancji w rozkładach normalnych).

Więcej przykładów modeli generujących można znaleźć np. w książce: `http://www.cs.ubc.ca/~murphyk/MLbook/`.

[1] Dla przejrzystości pominięto warunkowanie $\boldsymbol{\theta}$.

Modele dyskryminujące. Korzystanie z *modeli dyskryminujących* (ang. *discriminative modeling*) ma na celu modelowanie rozkładu warunkowego $p(y|\mathbf{x}, \boldsymbol{\theta})$, gdzie $\boldsymbol{\theta} \in \mathbb{R}^M$ oznacza wektor parametrów modelu. W wielu zastosowaniach interesuje nas wyłącznie znalezienie zależności klasy od zadanych cech, natomiast poznanie zależności dot. cech jest nieistotne. Co więcej, zwróćmy uwagę, że do podejmowania decyzji rozkład $p(y|\mathbf{x})$ jest wystarczający. Dlatego też podejście z modelowaniem wprost rozkładu y pod warunkiem $\mathbf{x}$ nazywa się *modelami dyskryminującymi*.

Przykładami modeli dyskryminujących są:

- *Logistic Regression*: przyjmujemy kombinację liniową cech, natomiast rozkład warunkowy modelowany jest za pomocą funkcji sigmoidalnej (przypadek dwuklasowy) lub softmax (przypadek wieloklasowy);
- κ-*Nearest Neighbor*: dla zadanej funkcji jądra, rozkład prawdopodobieństwa klasy pod warunkiem cech estymowany jest w sposób nieparametryczny przy użyciu wartości klas dla $\kappa \in \mathbb{N}$ najbliższych obiektów;
- *Sieć neuronowa, Multilayer Perceptron* (MLP): jeżeli przyjmujemy funkcję sigmoidalną lub softmax na wyjściu sieci, to wówczas MLP może być traktowany jako probabilistyczny model dyskryminujący.

Więcej o modelach dyskryminujących można znaleźć np. w książce: `http://www.cs.ubc.ca/~murphyk/MLbook/`.

Modele funkcyjne. Trzecim podejściem jest zaproponowanie *funkcji dyskryminującej* (ang. *discriminant function*), która nie ma interpretacji probabilistycznej, ale która pozwala na przyporządkowaniu wektorowi cech wartości klasy.

Przykładami modeli funkcyjnych są:

- *Sieć neuronowa, Multilayer Perceptron* (MLP): jeżeli przyjmujemy inną funkcję na wyjściu sieci niż funkcja sigmoidalna czy softmax (w konsekwencji należy wybrać inną funkcję celu uczenia niż funkcja wiarygodności), to wówczas MLP jest traktowana jako funkcja dyskryminująca;
- *Support Vector Machines* (SVM): jest to model dla przypadku dwuklasowego mający na celu znalezienie największego marginesu rozdzielającego klasy. W przypadku wieloklasowym należy wykorzystać K modeli i zastosować technikę predykcji 1 vs ALL.

Więcej o funkcjach dyskryminujących można znaleźć np. w książce: `http://www.cs.ubc.ca/~murphyk/MLbook/`.

Uczenie

Uczenie polega na estymacji parametrów modelu $\boldsymbol{\theta}$ przy użyciu danych (ciągu treningowego) $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ poprzez minimalizację zadanej funkcji celu (kryterium uczenia). W przypadku modeli probabilistycznych zazwyczaj przyjmuje się *ujemny logarytm funkcji wiarygodności* (ang. *negative log-likelihood*) lub, dla zadanego rozkładu *a priori* na parametry, *ujemny logarytm rozkładu a posteriori*.[2] Natomiast w przypadku modeli nieprobabilistycznych (tj. funkcji dyskryminujących) należy zaproponować innego rodzaju funkcję celu, np. dla klasyfikacji binarnej może to być błąd średniokwadratowy lub entropia krzyżowa (ang. *cross-entropy loss*).

Dodatkowo, w celu osiągnięcia zadanych własności modelu, do kryterium uczenia często dodaje się *regularyzację* (ang. *regularization*). Regularyzacja poprawia proces uczenia, np. poprzez przeciwdziałanie zbytniemu dopasowaniu modelu do danych (ang. *overfitting*) lub osiąganiu rozwiązań rzadkich (ang. *sparse representation*). Przykładem regularyzatora jest uwzględnienie normy ℓ_2 na parametry.

Czasem wartości parametrów minimalizujących kryterium uczenia przy zadanym ciągu obserwacji mogą być wyznaczone w postaci analitycznej (ang. *closed form*), jednakże dla większości użytecznych modeli nie jest to możliwe. Z tego powodu stosuje się numeryczne algorytmy optymalizacji. Do najczęściej stosowanych algorytmów w uczeniu maszynowym zalicza się metodę **gradientu prostego** (ang. *gradient descent*) lub jej wariant – metodę **stochastycznego gradientu prostego** (ang. *stochastic gradient descent*), w której uaktualnianie wag w pojedynczej iteracji odbywa się przy użyciu pojedynczej lub co najwyżej kilku obserwacji na raz, tzw. mini-paczek (*ang. mini-batch*). W celu stosowania takiego podejścia należy wyznaczyć gradient kryterium uczenia lub przynajmniej jego aproksymację.

Należy również dodać, że w przypadku niektórych modeli stosuje się dedykowane algorytmy uczenia, np. algorytm *Sequential Minimal Optimization* (SMO) dla SVM.

Więcej na temat uczenia modeli można znaleźć np. w książce: `http://www.cs.ubc.ca/~murphyk/MLbook/`.

[2] Standardowo metoda ta zwana jest *metodą maksymalnej a posteriori* (MAP), jednak dla spójności w niniejszym dokumencie mówimy o **minimalizacji** funkcji celu, dlatego należy uwzględnić znak minus.

Predykcja

Modele generujące. Po wyuczeniu modelu generującego, dla nowego obiektu $\mathbf{x}^{new}$ możemy wyznaczyć rozkład warunkowy $p(y|\mathbf{x}^{new})$ korzystając ze wzoru Bayesa:[3]

$$\begin{aligned} p(y|\mathbf{x}^{new}) &= \frac{p(\mathbf{x}^{new}|y)p(y)}{p(\mathbf{x}^{new})} \\ &= \frac{p(\mathbf{x}^{new}|y)p(y)}{\sum_{y'} p(\mathbf{x}^{new}|y')p(y')} \end{aligned}$$

Ostatecznie wartość klasy określana jest na podstawie wartości prawdopodobieństwa, tj. predykowana wartość klasy dla nowego obiektu przyjmuje największą wartość prawdopodobieństwa:

$$y^* = \arg\max_y p(y|\mathbf{x}^{new}). \tag{3}$$

Modele dyskryminujące. Po wyuczeniu modelu dyskryminującego, dla nowego obiektu dokonujemy predykcji analogicznie jak w przypadku modeli generujących z użyciem (3).

Modele funkcyjne. Funkcja dyskryminująca przyporządkowuje obiektowi wartość klasy. Dlatego po jej wyuczeniu, predykcja polega na poznaniu wartości funkcji dyskryminującej dla nowego obiektu.

Instrukcja wykonania zadania

1. Należy pobrać dane treningowe `train.pkl` ze strony kursu. Plik `train.pkl` zawiera dane obrazami w postaci macierzy $N \times D$, które mają być wykorzystane w procesie uczenia ($N = 60000$). Każdy z obrazów jest przekształcony do D-wymiarowego wektora, który stanowi odrębny wiersz macierzy z danymi. Obrazy są wielkości 36×36 pikseli ($D = 1296$). Aby zwizualizować wybrany wiersz `x` z macierzy X należy przekształcić go do obrazu poleceniem `np.reshape(x,(36,36))`, a następnie użyć polecenia `plt.imshow`.

   Plik `train.pkl` zawiera również N etykiet klas dla obrazów, które przyjmują wartości ze zbioru $\{0, \ldots, 9\}$ (wartość etykiety odpowiada typowi garderoby, które reprezentuje zdjęcie).

2. Posiłkując się otrzymanymi danymi treningowymi, należy wyuczyć model, który następnie wykorzystany będzie do klasyfikacji obiektów na obrazach. Ewentualny ciąg walidacyjny należy wyróżnić samodzielnie.

3. W celu klasyfikacji obrazów należy uzupełnić funkcję `predict` w pliku `predict.py` w taki sposób, żeby funkcja dla macierzy wejściowej $\mathbf{X}$, w której poszczególne wiersze zawierają

[3] Dla przejrzystości pominięto warunkowanie $\boldsymbol{\theta}$.

obrazy do sklasyfikowania, uzupełniła wektor $\mathbf{y}$ o przewidywane etykiety klas (i-ty wiersz $\mathbf{X}$ odpowiada i-temu elementowi wektora $\mathbf{y}$).

4. Jakość predykcji zaproponowanego modelu można sprawdzić raz dziennie poprzez załadowanie pliku `predict.py` plus ew. plików pomocniczych w postaci paczki `zip` na serwer:

   `http://mlg.ii.pwr.edu.pl/msid/`

   Przeprowadzona zostanie wówczas weryfikacja jakości opracowanego modelu na podstawie 25% danych testowych.

5. Wielkość paczki `zip` nie może przekraczać 5 MB. Czas wykonania funkcji `predict` nie może przekraczać 1 minuty dla 2500 przykładów (25% zbioru testowego) oraz 4 minut dla 10000 przykładów (cały zbiór testowy). Dodatkowo plik `predict.py` musi znajdować się "na wierzchu" po rozpakowaniu paczki `zip`, a nie w dodatkowych podkatalogach.

6. Serwer wykonuje ewaluację programów codziennie o godzinie 00:00 z użyciem Python 3.6 z pakietem numpy. Należy zapewnić, aby wgrywany program był zgodny z tym standardem.

7. Gotowy program należy wgrać na serwer przed terminem określonym w zadasach zaliczenia kursu. Ostateczny wynik, pozycja rankingowa oraz ocena zostanie wyliczona na podstawie całego zbioru testowego po tym terminie. Do oceny brany jest pod uwagę tylko ostatni program załadowany na serwer.

8. Aby zaliczyć kurs na ocenę pozytywną, jakość predykcji programu musi być wyższa niż 50% na całym zbiorze testowym. Należy wziąć pod uwagę, że jakość predykcji na 25% zbioru może różnić się nawet o 1% w stosunku do jakości na całym zbiorze.