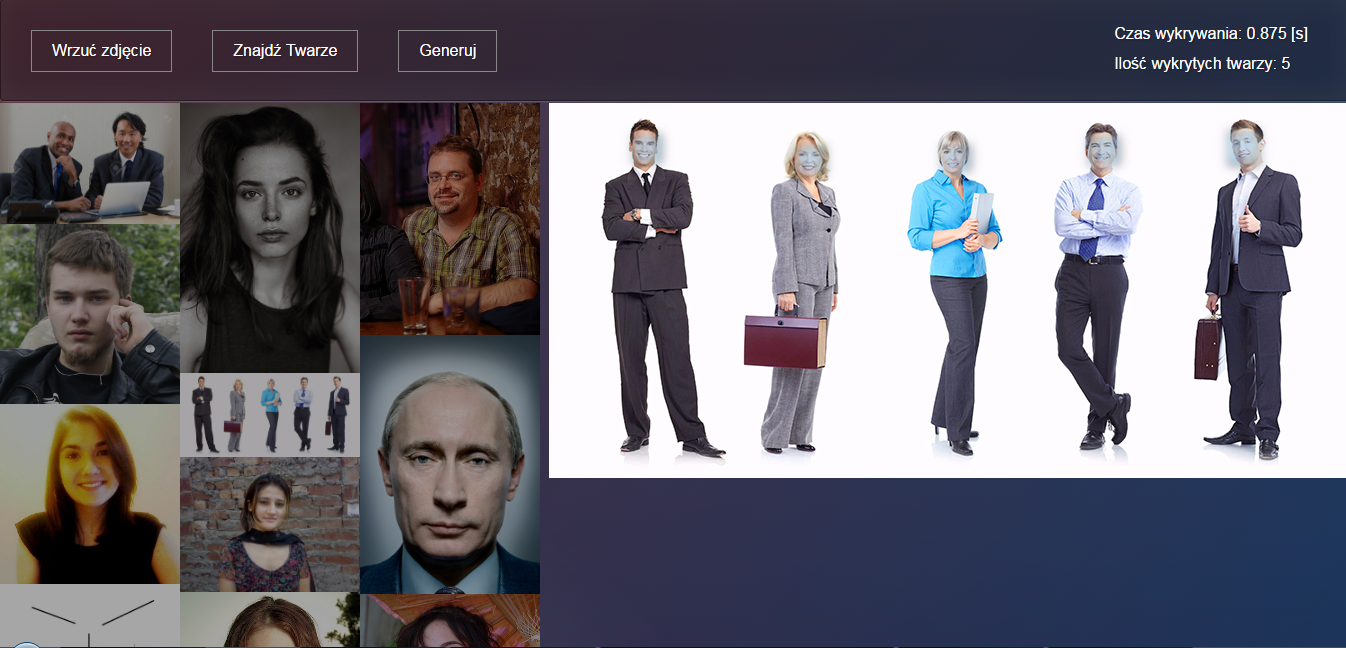
Dawid Sopata

Opis projektu z przedmiotu Podstawy Sztucznej Inteligencji

Celem mojego projektu było wykonanie aplikacji webowej, umożliwiającej detekcje twarzy ze zdjęcia, następnie wygenerowanie rozmycia, oraz przygotowanie pliku do pobrania.



Sam kod biblioteki pobrałem stąd:

https://github.com/jaysalvat/jquery.facedetection

Biblioteka opiera się o algorytm YEF (Yet Even Faster) Real-Time Object Detection.

Jak działa algorytm?

Algorytm się opiera o punkty kontrolne (ang. Control point).

Bardzo podstawowy instrument stosowany w algorytmie nazywamy funkcję punktu kontrolnego (przemianowany na jasności funkcji binarnej do refleksji, wdrożenie w CCV działa tylko na wartości jasności). Dla danego obrazu regionu WXH jedna funkcja składa się z dwóch zbiorów punktów kontrolnych w [1] w [2] ... A [n] i B [1], B [2], ..., b [m]. Aby zakwalifikować dany obszar obrazu, funkcja analizuje wartości pikseli w punktach kontrolnych w grupie A i grupa B dla odpowiednich obrazów (w oryginalnym rozmiarze, połowie rozmiaru i ćwierć-rozmiaru). Funkcja zwraca "tak", jeśli wszystkie wartości pikseli w grupie A jest większa / mniejsza niż jakiekolwiek wartości pikseli w grupie B.

Val (x)> Val (y)

Program bbfcreate stworzy kilka silnych klasyfikatorów liniowych z funkcji punktu kontrolnego z wykorzystaniem adaboost.

*„W skrócie, AdaBoost działa w ten sposób, że w kolejnych iteracjach trenuje a następnie mierzy błąd wszystkich dostępnych słabych klasyfikatorów. W każdej następnej iteracji "ważność" źle zakwalifikowanych obserwacji jest zwiększana, tak że klasyfikatory zwracają na nie większą uwagę.”*

Źródło: wikipedia

Boosting jest ogólną metodą służącą zwiększeniu skuteczności dowolnego algorytmu uczenia. Idea Budowanie “mocnego i złożonego klasyfikatora” ze “słabych i prostych klasyfikatorów”.

Leslie Valiant i Michael Kearns byli pierwszymi, którzy pokazali, że “słabe” algorytmy uczące, których skuteczność jest nawet niewiele lepsza niż losowe zgadywanie, mogą być wykorzystane do stworzenia dowolnie skutecznego “silnego” klasyfikatora.

* Robert Schapire jako pierwszy przedstawił algorytm wzmacniania działający w czasie wielomianowym.
* Yoav Freund zaproponował znacznie bardziej efektywny algorytm wzmacniania, który mimo tego, że był w pewnym sensie optymalny, miał pewne istotne w praktyce wady.
* Pierwsze eksperymenty z tymi wczesnymi algorytmami wzmacniania zostały przeprowadzone przez zespół Drucker, Schapire, Simard i dotyczyły zadania OCR (ang. optical character recognition), w którym sieci neuronowe były użyte jako “proste” klasyfikatory.
* Freund i Schapire przedstawili algorytm AdaBoost, który rozwiązał wiele praktycznych trudności wcześniejszych algorytmów wzmacniania.

**11.1.2. AdaBoost - opis algorytmu**

* Algorytm na wejściu otrzymuje zbiór treningowy (x_{1},y_{1}),(x_{2},y_{2}),\ldots,(x_{m},y_{m}), gdzie każdy x_{i} należy do pewnej dziedziny problemu X, natomiast każda etykieta (decyzja) y_{i} należy do pewnego zbioru Y. Dla ułatwienia będziemy na razie zakładać, że Y=\{-1,+1\}.
* AdaBoost (Adaptive Boosting) wywołuje wybrany “słaby” algorytm uczący w serii T iteracji. Zakładamy, że błąd uzyskiwanych klasyfikatorów na zbiorze treningowym jest mniejszy niż \frac{1}{2}.
* Jedną z głównych idei algorytmu jest strojenie rozkładu (lub wag elementów) dla zbioru treningowego. Wagę i-tego elementu ze zbioru treningowego w iteracji t będziemy oznaczali przez D_{t}(i).
* Początkowo wszystkie wagi są ustawione na równe wartości;
* Po każdej iteracji, wagi elementów źle klasyfikowanych są zwiększane. Dzięki temu mamy możliwość skierowania uwagi “słabego” klasyfikatora na pewne elementy (trudne do wyuczenia) ze zbioru treningowego.
* Zadaniem “słabego” algorytmu uczącego jest zbudowanie klasyfikatora (ang. hypothesis) h_{t}:X\rightarrow Y odpowiedniego dla aktualnego rozkładu D_{t}.
* Skuteczność takiego klasyfikatora jest mierzona przez jego błąd (z uwzględnieniem rozkładu D_{t}):

|  |  |  |
| --- | --- | --- |
|  | \varepsilon _{t}=Pr_{{D_{t}}}[h_{t}(x_{i})\neq y_{i}]=\sum _{{i:h_{t}(i)\neq y_{i}}}D_{t}(i) |  |

* W praktyce “słaby” algorytm uczący może być algorytmem, który uwzględnia rozkład D_{t}. Nie jest to jednak konieczne. Kiedy algorytm nie pozwala na bezpośrednie uwzględnienie rozkładu D_{t}, losuje się (względem D_{t}) podzbiór zbioru treningowego, na którym następnie wywołuje się algorytm uczący.
* Kiedy AdaBoost dostaje klasyfikator h_{t} dobierany jest parametr \alpha _{t}. Intuicyjnie \alpha _{t} odpowiada za wagę jaką przykładamy do klasyfikatora h_{t}. Zauważmy, że \alpha _{t}\geq 0 gdy \varepsilon _{t}\leq\frac{1}{2}. Ponadto \alpha _{t} rośnie kiedy \varepsilon _{t} maleje.
* Rozkład D_{t} jest następnie zmieniany tak, aby zwiększyć (zmniejszyć) wagi elementów zbioru treningowego, które są źle (dobrze) klasyfikowane przez h_{t}. Stąd wagi mają tendencję do skupiania się na “trudnych” przykładach.

Dane: (x_{1},y_{1}),\ldots,(x_{m},y_{m}), gdzie x_{i}\in X, y_{i}\in Y=\{-1,+1\}   
Inicjalizacja: D_{1}(i)=\frac{1}{m}\  dla \  i=1,\ldots,m   
   
for t=1,\ldots,T do:

* <+-> Wykorzystując “słaby” algorytm uczący zbuduj klasyfikator h_{t}:X\rightarrow\{-1,+1\}

(uwzględniając rozkład D_{t}).

* <+-> \varepsilon _{t}=Pr_{{D_{t}}}[h_{t}(x_{i})\neq y_{i}]=\sum _{{i:h_{t}(i)\neq y_{i}}}D_{t}(i)
* <+-> \alpha _{t}=\frac{1}{2}\ln(\frac{1-\varepsilon _{t}}{\varepsilon _{t}})
* <+-> Uaktualnij wagi elementów zbioru treningowego:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | \displaystyle D_{{t+1}}(i) | \displaystyle= | \displaystyle\frac{D_{t}(i)}{Z_{t}}\times\left\{\begin{array}[]{l}e^{{-\alpha _{t}}}\ \textrm{ jeśli }h_{t}(x_{i})=y_{i}\\ e^{{\alpha _{t}}}\ \ \textrm{ jeśli }h_{t}(x_{i})\neq y_{i}\end{array}\right. |  |
|  |  | \displaystyle= | \displaystyle\frac{1}{Z_{t}}\times D_{t}(i)\times e^{{-\alpha _{t}y_{i}h_{t}(x_{i})}} |  |

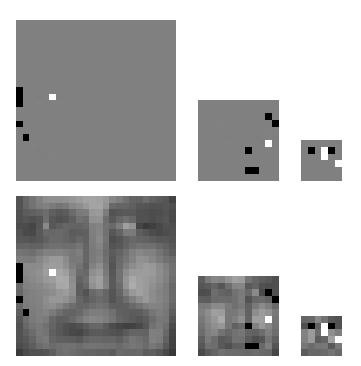
* Z_{t} jest czynnikiem normalizującym (wybranym tak, aby D_{{t+1}} było rozkładem).

Wynikowy klasyfikator powstaje za pomocą ważonego głosowania klasyfikatorów h_{t}, gdzie \alpha _{t} jest wagą przypisaną klasyfikatorowi h_{t}.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| t= | 1 | 2 | \dots | T |
| słabe klasyfikatory | h_{1} | h_{2} | \dots | h_{T} |
| wagi | \alpha _{1} | \alpha _{2} | \dots | \alpha _{T} |
|  |  |  |  |  |

Źródło opisu algorytmu adaboost: <http://mst.mimuw.edu.pl/lecture.php?lecture=syd&part=Ch11#S1.SS2>

Funkcja punktu kontrolnego jest na tyle prosta, że po wygenerowaniu piramidy obrazu (seria obrazów, które są zmniejszone z oryginalnego obrazu w rozmiarze WXH do W / 2xH / 2, W / 4xH / 4 ...), nie powodu do dalszego przetwarzania obrazu.



**Przykład piramidy obrazu**

Algorytm osiąga podobną dokładność (82.97% z 12 fałszywych alarmów VS 86.69% z 15 fałszywych alarmów – porównując z implementacją CCV w C).

Porównując z domyślną metodą OpenCV ten wykrywacz twarzy, działa 3 razy szybciej.

s