Dokumentacja projektu z przedmiotu PSZT

“Klasyfikator oparty o SVM w rozpoznawaniu cyfr”

Paweł Rączkowski i Paweł Prusałowicz

Opiekun projektu: mgr inż. Mikołaj Markiewicz

1. Treść zadania

Zaimplementować klasyfikator oparty na modelu maszyny wektorów nośnych (SVM) w wersji liniowej i nieliniowej (kernel: poly/rbf/sigmoid). Porównać wyniki dla strategii one-vs-one i one-vs-all do klasyfikacji zbioru ręcznie pisanych cyfr. Zbiór danych do użycia: MNIST - http://yann.lecun.com/exdb/mnist/. Uzyskane rezultaty porównać z wynikami dla wybranej implementacji algorytmu ML z dostępnych bibliotek np. Scikit-learn, WEKA, MLlib, Tensorflow/Keras etc.

2. Przyjęte założenia, doprecyzowanie treści

Każdy obrazek dostarczany był do modelu w formie wektora o wymiarze 28x28, gdyż każdy taki obrazek jest zbiorem pikseli o wymiarach 28x28. Każdy element wektora reprezentował jasność danego piksela w skali: 0- biały; 255- czarny.
Bias, czyli skalar b w równaniu hiperpłaszczyzny w SVM w wersji liniowej był obliczany dla jednego modelu jako średni błąd jaki model zrobił na danych trenujących.
Pewność predykcji dokonywanej przez modele dla danego przykładu była liczona jako (**w***x+b)/norm(**w**) czyli odległość wektora **x** od granicy decyzyjnej.
W strategii one vs all kiedy próbuje się rozróżnić np. cyfrę 0 od reszty to etykiety, które pierwotnie należą do zbioru {0,1,2,3,4,5,6,7,8,9} są konwertowane w ten sposób: ten przykład, któremu przypisana jest cyfra 0 zmieniamy etykietę na 1, a w reszcie przypadków na -1.
W strategii one vs one, kiedy model uczy się rozróżniać przykładowo 0 od 1, to przykłady którym przypisana była etykieta 0 zamieniamy na 1, a etykieta 1 zamieniamy na -1 i reszty przykładów nie bierzemy do naszego modelu.

3. Podział odpowiedzialności

Paweł Rączkowski: przygotowanie danych do modelu one vs all oraz one vs one i implementacja liniowej wersji SVM
Paweł Prusałowicz: implementacja nieliniowej wersji SVM i testowanie algorytmów ML z biblioteki Scikit-Learn.

4. Opis algorytmu i uzasadnienie sposobu realizacji

Na początku działania algorytmu wczytywane są dane do trenowania i do testowania modelu, czyli wektory pikseli oraz etykiety(labele) które określają jaką cyfrę reprezentuje dany zbiór pikseli. Następnie, z racji tego, że elementy mają wartości z zakresu 0-255 wszystkie poddawane są normalizacji [0;1], wartość każdego elementu dzielona jest przez 255. Po normalizacji inicjalizuje się wektor wag o takim samym rozmiarze jak wektor pikseli i ustala się jego elementy na 0. Po wstępnym przygotowaniu danych, użytkownik wybiera opcję którą chce przetestować(np. model liniowy w taktyce one-vs-all).

W przypadku taktyki one-vs-all tworzonych jest 10 modeli: model, który odróżnia 0 od reszty; 1 od reszty; 2 od reszty itd. Dlatego na początku wektor etykiet ze zbioru {0,1,2,3,4,5,6,7,8,9} konwertowany jest na {1,-1} w taki sposób, że jeżeli budujemy model do odróżniania 0 od reszty to przykłady, którym przypisana była etykieta 0 to zastępujemy 1 a

reszta przykładów ma indeks -1. Następnie wykonywane jest trenowanie 10 modeli za pomocą algorytmu trenującego.

Algorytm trenujący **modele liniowe** przebiega następująco:

1) jako argumenty przyjmuje zainicjalizowane wagi, parametr alfa- tzw.learning rate, przykłady do trenowania czyli wektory pikseli (wszystkie obrazki z training setu) oraz przekonwertowane wcześniej labele. Ustawienie bias=0
2) Dla każdego przykładu w danych do trenowania:
    a) liczona jest suma iloczynu skalarnego wag i biasu
    b) jeżeli **iloczyn** tej sumy z odpowiadającą temu przykładowi etykiecie jest większy/równy 1 to znaczy, że przykład jest dobrze zaklasyfikowany i aktualizujemy wagi: **w=w-2***alfa*C***w** gdzie C jest wybierany eksperymentalnie ale u nas ma wartość 1/10000 albo 1/100000
    c) w przeciwnym wypadku jest zła klasyfikacja: aktualizujemy **w=w+alfa*wartość_etykiety*[x-2***alfa*C***w]**
    bias+=(1-**iloczyn**)/N gdzie N to liczba przykładów
3) Powyższe punkty są wykonywane dwukrotnie gdyż pojedyncze przejście po danych było niewystarczające. Algorytm zwraca wytrenowane wagi (**w)** i bias.

Dla **modelu nieliniowego** skorzystano z uproszczonego algorytmu SMO opisanego w http://cs229.stanford.edu/materials/smo.pdf. Funkcja klasyfikująca przyjmuje postać:
f(x) = □ sum(α_i * y(i) *⟨x(i), x⟩) + b,
⟨x(i), x⟩ oznacza Kernel, przyjęto funkcję jądra kwadratowego
Poszukiwane są takie parametry α_i, które spełniają założenia Kuhna-Tuckera(KKT), czyli warunki konieczne dla lokalnego rozwiązania problemu optymalizacyjnego:

- αi =0 ⇒ y(i)(wTx(i) +b)≥1
- αi=C ⇒ y(i)(wTx(i)+b)≤1
- 0<αi <C ⇒ y(i)(wTx(i) +b)=1

Algorytm pobiera dwa parametry α - α_i oraz α_j (iteracja po α_i, i = 0,...,m , gdzie m jest długością wektora danych trenujących oraz losowe dobieranie α_j) oraz wzajemnie optymalizuje ich wartości tak długo, aż parametry α nie zbiegną do optymalnego rozwiązania. Bazując na wyznaczonych parametrach, ustala bias b modelu.

Po wytrenowaniu następuje walidacja każdego z modeli na danych testujących- pewność każdego modelu dla danego przykładu jest liczone jako odległość punktu od granicy decyzyjnej ((**w*x**+b/norm(**w)**). Model, który zyskuje największą wartość walidacji jest wybierany jako predykcja modelu.

W przypadku **one vs one** tworzone jest 45 modeli, gdyż każdy model uczy się odróżniać jedną cyfrę od innej. Następnie wykonywany jest algorytm trenujący na każdym z 45 modeli. Walidacja przebiega tak, że konfrontujemy każdą cyfrę z każdą czyli 0 z 1; 0 z 2; 0 z 3 itd. i w każdej takiej konfrontacji jeżeli model przewiduje daną cyfrę to naliczane są jej punkty (dodając także do tego pewność predykcji jak w modelu one vs all). Cyfra z największą liczbą punktów jest wybierana jako predykcja do przykładu.

5. Raport z przeprowadzonych eksperymentów

Czas budowania modeli oraz dokładność predykcji:

- Implementacja liniowa w wersji one vs all dla alpha=0.01 i C=1/10000:

Dokładność: **88.72%**, Czas: 0:18:10.693929

- Implementacja liniowa w wersji w wersji one vs one:

Dokładność: **69.53%**, Czas: 0:18:55.458439

- Implementacja nieliniowa one vs all (C=1.0, epsilon=0.001):

Dokładność: **39.0%**, Czas: 3:20:30.534534

- Implementacja nieliniowa one vs one (C=1.0, epsilon=0.001):

Dokładność: **20.5%**, Czas: 0:51:23.352356

- Biblioteka Sklearn - Implementacja liniowa one vs all (C=0.1, gamma=0.01):

Dokładność: **94.73%**, Czas = 0:04:24.092215

- Biblioteka Sklearn - Implementacja liniowa one vs one (C=0.1, gamma=0.01):

Dokładność: **94.73%**, Czas = 0:04:26.572395

- Biblioteka Sklearn - Nieliniowy, kernel= 'poly' one vs all (C=0.1, gamma=0.01):

Dokładność: **97.44%**, Czas = 0:03:54.037982

- Biblioteka Sklearn - Implementacja liniowa one vs one (C=0.1, gamma=0.01):

Dokładność: **97.44%**, Czas = 0:03:55.339451

6. Wnioski

Implementacja modelu liniowego ma wysoką dokładność, jednak czas budowania modelu wymaga optymalizacji (jest ponad 4-krotnie dłuższy od odpowiadającego czasu wykonania przez bibliotekę sklearn).
Z kolei dokładność zaimplementowanego modelu nieliniowego na podstawie uproszczonego algorytmu SMO jest niska z uwagi na możliwość wykonania **jedynie jednej iteracji algorytmu w akceptowalnym czasie**. Złożoność obliczeniowa jest proporcjonalna do wielkości zestawu danych treningowych, a przy konieczności rozważenia w iteracji każdego z 60000 parametrów α (60000 - długość wektora danych oraz w związku z tym wektora parametrów α), oraz bardzo częstym wykonaniu operacji numpy.dot na wektorach tej długości, czas wykonania algorytmu jest bardzo długi.
Do poprawy wyniku mogłoby prowadzić wykonanie większej liczby iteracji lub zmniejszenie zestawu danych treningowych (na przykład 10-krotnie). Także wykorzystanie pełnego algorytmu SMO mogłoby zoptymalizować czas trenowania modelu oraz gwarantowałoby jego zbieżność.

7. Instrukcja użytkownika

Program uruchomiony może zostać poprzez wykonanie komendy **python main.py** w terminalu z poziomu folderu, w którym znajduje się projekt. Dla wykonania trenowania modelu oraz predykcji na podstawie danych testowych wystarczy wybrać numer numer reprezentujący dany model (opis dostępny przy starcie programu) oraz wcisnąć enter.
Aby uruchomić modele z bibliotek służące do porównania (opisane numerami 5-8), konieczna jest instalacja biblioteki Sklearn.