Algorytm indukcji klasyfikatora za pomocą EA z automatycznym przełączaniem ukierunkowań

Anna Manerowska, Michal Kozakiewicz

20.01.2009

1 Wstęp

Jako projekt na przedmiot MEUM (Metody Ewolucyjne Uczenia Maszyn) dostaliśmy za zadanie stworzyć klasyfikator rozwijany za pomocą algorytmu ewolucyjnego, którego ukierunkowanie zmieniałoby się podczas uruchomienia. EA zostanie użyty do zmian drzew decyzyjnych. Zmienne ukierunkowanie rozwoju drzew uzyskamy przez modyfikacje funkcji celu oraz operatora mutacji w trakcie uruchomienia algorytmu.

2 Opis problemu klasyfikacji

Zadaniem klasyfikacji będzie zakwalifikowanie instrumentu giełdowego do jednej z trzech grup: [kup, sprzedaj, nie ruszaj].

- kup - algorytm uważa, że cena danego instrumentu wzrośnie
- sprzedaj - algorytm uważa, że cena danego instrumentu zmaleje
- nie ruszaj - algorytm nie potrafi zdecydować

Na podstawie ciągu cen oraz ustalonego maksymalnego czasu posiadania aktyw, wyznaczamy grupę do której należy dany ciąg;

3 Dane

Klasyfikację będziemy przeprowadzać na cenach archiwalnych wybranych instrumentów. W projekcie zdecydowaliśmy się pracować z danymi typu:

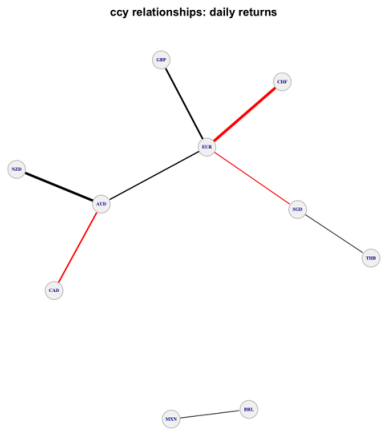
- Daily trading - czyli transakcje na przestrzeni dni/tygodni; Wymagana jest jedynie cena zamknięcia danego rynku.

Wybraliśmy ten rodzaj cen ze względu na łatwość zdobycia danych archiwalnych. Jako źródło użyte zostało archiwum Oanda[14].

Wektorem wejściowym jest ciąg cen z pewnego okresu czasu.

Do analizy wybraliśmy pary walut:

- AUD/EUR
- BRL/EUR
- CAD/EUR
- CHF/EUR
- GBP/EUR
- MXN/EUR
- NZD/EUR
- SGD/EUR



Rysunek 1: Korelacja między walutami

Niektóre waluty mają ze sobą silną (grubość krawędzi) korelację (oznaczone kolorem czarnym) lub anty-korelację (czerwone), inne mają słabe związki Rys. 1.

Na podstawie [7] wybraliśmy pary walut które są ze sobą skorelowane lub nie, aby zbadać zachowanie algorytmu w obu przypadkach.

3.1 Analiza

Aby zakwalifikować ciąg cen do jednej z trzech grup, potrzebne są dane testowe określające odpowiednią grupę dla danego ciągu cen. Aby wygenerować takie dane użyliśmy następującego sposobu:

- wybieramy cene C oddaloną o ilość cen użytych do klasyfikacji od pierwszej ceny w zbiorze
- przeszukujemy ceny występujące po C w zakresie określonym przez parametr maksymalnego czasu trzymania otwartej pozycji (w dniach):
  - szukamy maksymalnej i minimalnej ceny w tym podzbiorze;
  - jeżeli cena minimalna jest mniejsza o daną wielkość (spread) od obecnej ceny, klasyfikujemy jako SPRZEDAJ;
  - jeżeli cena maksymalna jest większa o daną wielkość (spread) od obecnej ceny, klasyfikujemy jako KUP;
  - klasyfikujemy jako NIE RUSZAJ;
- przechodzimy o jedną cenę do przodu i generujemy kolejną parę ciąg, klasa;

4 Narzędzia realizacji

Projekt został wykonany w Javie, przy użyciu następujących bibliotek:

- Watchmaker [12] - biblioteka do metod ewolucyjnych

Wybór języka kompilowanego, jest uzasadniony dużymi wymaganiami obliczeniowymi metod ewolucyjnych; Javy, ze względu na dostępność bibliotek i łatwość programowania.

Zrezygnowaliśmy z prototypu w R, ze względu na łatwą możliwość wykonania go w Javie przy jednoczesnej oszczędności czasu.

5 Opis algorytmu

Działanie algorytmu wygląda następująco:

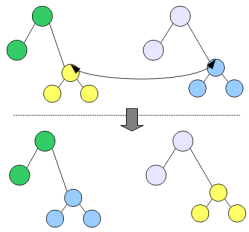
- populację bazową budujemy przy użyciu drzew losowych o zadanej wielkości maksymalnej
- osobniki dokonują klasyfikacji
- oceniamy podstawową sprawność osobników na podstawie funkcji celu F

- funkcja celu przyjmuje jako dodatkowy argument modyfikator sprawności, który zmienia sprawność osobnika
- wybieramy e osobników elitarnych, które zostaną dodane do populacji wyjściowej bez zmian
- z pośród całej populacji wybieramy daną metodą ruletkową p osobników, gdzie p = wielkość populacji - e;
- stosujemy operatory ewolucyjne na otrzymanej populacji
- na poziomie mutacji również możliwe jest dodanie modyfikatora, który określa gdzie w wybranym osobniku zachodzi mutacja
- proces powtarzamy do momentu gdy osiągniemy jeden z warunków końcowych

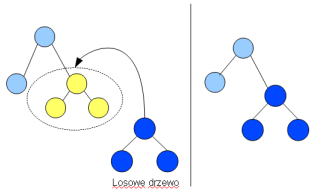
5.1 Krzyżowanie

Krzyżowanie dokonywane jest w jednym punkcie, generując dwa nowe osobniki z dwóch osobników wejściowych. Przebiega w następujący sposób:

- wybór dwóch rodziców
- wybór indeksu pod-drzewa rodzica 1
- wybór indeksu pod-drzewa rodzica 2
- zamiana wybranych pod-drzew w obu rodzicach
- zwrócenie zmodyfikowanych rodziców do puli



Rysunek 2: Krzyżowanie



Rysunek 3: Mutacja poddrzewa

5.2 Mutacja

Operator mutacji dla drzew decyzyjnych wstawia we wskazanym miejscu nowo wygenerowane pod-drzewo losowe rys.3

Poziom drzewa na którym następuje mutacja, może być zmieniany podczas uruchomienia algorytmu. Służy do tego modyfikator mutacji. Zaimplementowane modyfikatory to:

- ConstModifier - nie modyfikuje indeksu dziecka;
- RangeModifier - modyfikuje indeks dziecka na podstawie numeru generacji; Jako parametry przyjmuje zasięgi numerów generacji

oraz wielkość o jaką zwiększany jest indeks: np. [0,100,1] oznacza, że dla generacji od 0 do 100 indeks zwiększany jest o jeden, w praktyce mutowane jest lewe pod-drzewo;

5.3 Upraszczanie

Ponieważ algorytm nie wprowadza żadnych ograniczeń do wyboru argumentu w danym węźle decyzyjnym, możliwa są dwie sytuacje które zmniejszają efektywność i czytelność generowanych drzew:

- węzeł decyzyjny o dwóch takich samych klasach;
- węzeł decyzyjny o parametrze występującym juz w którymś z rodziców;

Aby wyeliminować te sytuacje wprowadzony został operator upraszczania. Podczas pisania dokumentacji powstał pomysł użycia informacji o duplikatach w celu obniżenia sprawności osobnika, szczególnie jeżeli klasyfikacja powtarzającego się parametru daje inne rezultaty. Częściowo jest to już zaimplementowane, można osiągnąć zbliżony efekt przez usunięcie upraszczania i zastosowanie modyfikatora karzącego duplikaty: ArugmentDuplicationModifier, z odpowiednimi wagami.

5.4 Funkcja celu

Funkcja celu F będzie zmieniać się w trakcie działania algorytmu. Jako podstawę sprawności wygenerowanego osobnika uznamy ilość poprawnych klasyfikacji. Dodatkowo wprowadziliśmy modyfikatory funkcji celu:

- NoOpModifier - nie modyfikuje funkcji celu;
- FunctionModifier - modyfikuje funkcję celu o wartość obliczoną przez dowolną funkcję przyjmującą jako parametr dane o populacji oraz osobnika testowanego:
  - testowana były funkcje okresowe np. sin
  - funkcje ze znaną asymptotą poziomą (np. arctan);
- EnsembleModifier - modyfikuje funkcję celu na podstawie przydatności osobnika dla całej populacji;
- ArgumentDuplicationModifier - ponieważ generowane drzewa decyzyjne mogą mieć powtarzające się parametry, dzięki temu operatorowi możemy karać lub nagradzać osobniki za ponowne wykorzystanie parametrów;

6 Instalacja i uruchomienie

Projekt można pobrać ze strony:

http://code.google.com/p/meum/

Do kompilacji wymagany jest ANT.

Aby uruchomić testy, należy wywołać komendę:

ant tests

Wygenerowany zostaje katalog tests, zawierający

- raport w postaci html, z wynikami oraz parametrami testu;
- obrazki drzew wygenerowanych podczas testu; Obrazki potencjalnie bardzo duże nie są zapisywane, ze względu możliwe przekroczenie pamięci uruchomieniowej;
- tabele podsumowujące uruchomione testy;

Ze względu na mnogość testów, nie są one zamieszczone w tym dokumencie. Ponieważ uruchomienie wszystkich testów zajmuje dużą ilość czasu (nawet do 25 min) na procesorze Intel Quad Core Q6600, przykładowy raport jest załączony do dokumentacji.

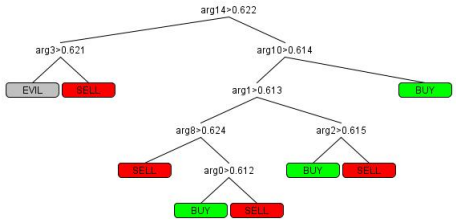
7 Format wyników

Wyniki podzielone są na cztery grupy, z czego dla nas interesujące są trzy:

- Fitness modifier tests - zawiera testy z różnymi modyfikatorami funkcji celu;
- Mutation modifier tests - zawiera testy z różnymi modyfikatorami mutacji;
- Combined tests - zawiera testy z oboma typami modyfikatorów;

Testy były projektowane w taki sposób aby sprawdzić zachowanie algorytmu z różnymi ustawieniami, jednocześnie automatyzując generowanie raportów. Każdy test wyświetla swoją konfigurację:

- nazwę;
- maksymalną głębokość generowanych poddrzew;
- wielkość populacji bazowej;
- ilość osobników elitarnych;
- użytą funkcję celu i jej modyfikatory;
- operatory genetyczne;
- obserwatorów ewolucji - obiekty zainteresowane informacjami o zmianach populacji;



Rysunek 4: Przykład drzewa wygenerowanego przez algorytm

- klasę generatora liczb losowych, użytego do testu;
- strategię selekcji;

Drzewo wynikowe przedstawione jest w postaci:

$$arg0 > 1.00?BUY : SELL \tag{1}$$

Generowany jest również obrazek drzewa w katalogu "images", nazwa testu odpowiada nazwie obrazka. Test wyświetla jego "podstawową"(bez użycia modyfikatorów) sprawność na zbiorze treningowym i testowym. Wartości są z przedziału 0-1 gdzie 1 oznacza 100% błąd klasyfikacji a 0 zakwalifikowanie wszystkich przykładów prawidłowo.

We wszystkich przypadkach ewolucja trwa 1000 generacji.

Mnogość parametrów bardzo utrudnia przeprowadzenie testów dostosowujących je wszystkie. Ponieważ tematem tego projektu były zmiany ukierunkowania podczas uruchomienia algorytmu, najwięcej testów przeznaczonych jest badaniu modyfikatorów tych ukierunkowań.

Podsumowanie działania dla danej grupy testów znajduje się w katalogu summary.

8 Omówienie rezultatów

Do omówienia rezultatów użyjemy załączonego raportu 0. Wymagany spread dla wszystkich testów wynosił 0.002, co dla większości rynków FOREX sprowadza się do 20 punktów (zmiany do 4 miejsc po przecinku). Innymi słowy na każdej zainwestowanej złotówce tracimy lub zyskujemy 20 zł, pomijając koszty transakcji. Spread ten jest duży ale dzięki takiemu wyborowi rośnie pewność, że klasyfikacja skończy się zyskiem, odrzucając mniejsze wahania ceny.

8.1 Modyfikatory mutacji

Modyfikatory mutacji testowane były ze standardową funkcją celu, na zbiorach treningowych z cenami od 01/09/2009 do 01/01/2010 oraz zbiorach testowych z cenami 02/01/2010 do 01/25/2010;

Klasyfikacja odbywała się na podstawie 15 poprzednich cen. Pozycja mogła być trzymana przez następne 15 dni.

8.1.1 ConstModifier

Wykonane zostały 3 testy bez zmian mutacji. Uzyskane wyniki są bardzo dobre dla zbioru treningowego ale bardzo złe dla zbioru testowego. Wygenerowane drzewa są wysokości 8,10,10. Nie używają całej puli parametrów. Dla rynku CAD/EUR algorytmowi nie udało się zakwalifikować poprawnie żadnego z przykładów zbioru testowego.

8.1.2 RangeModifier

Modyfikator ten bardzo wpływa na wysokość generowanych drzew, szczególnie w przypadku gdy nie używamy operatora upraszczania. Ponowne wylosowanie tego samego parametru ze zbioru o wielkości 15, jest bardzo prawdopodobne podczas 1000 generacji ewolucji 1000 osobników. Próbowaliśmy dobrać wartości zakresów w taki sposób aby:

- w miarę rozwoju populacji mutować co raz głębiej w drzewie, zakładając, że węzły bliżej korzenia dają już dobre efekty;
- sytuację odwrotną, aby uzyskać lepsze przechodzenie minimów lokalnych;
- strategie mieszaną, np na początku znaleźć najlepszy obszar przyciągania, i tam przeprowadzać dalszą ewolucję

Z przetestowanych par walut, te konfiguracje sprawują się dostatecznie dla CAD/EUR (0.22). Wygenerowane drzewa nie są olbrzymie, wysokość od 7 do 11.

8.1.3 Podsumowanie modyfikatorów mutacji

Zastosowanie modyfikatora zmieniającego miejsce mutacji ma bardzo duży wpływ na generowane drzewa, ale nie jest on odczuwalny gdy używamy operatora upraszczania przy małej ilości atrybutów. Modyfikator pozwala na dostarczenie do populacji zarówno nowych osobników jak i subtelne zmiany istniejących.

8.2 Modyfikatory funkcji celu

Modyfikatory funkcji celu testowane były ze standardową mutacją, na zbiorach treningowych z cenami od 01/09/2009 do 01/01/2010 oraz zbiorach testowych z cenami 02/01/2010 do 01/25/2010;

Klasyfikacja odbywała się na podstawie 15 poprzednich cen. Pozycja mogła być trzymana przez następne 15 dni.

8.3 FunctionModifier

W testach zastosowaliśmy następujące funkcje zmieniające ocenę osobnika:

$$abs(sin(depth * 0.1)) \tag{2}$$

$$-depth * 0.001 \tag{3}$$

$$arctan(depth * 0.1)) \tag{4}$$

$$depth/generation : generation > 0 \tag{5}$$

Ciekawym jest to, że okresowa funkcja 2) generuje bardzo małe drzewa, pomimo tego, że wielkość o którą zmienia funkcję celu jest mała, szczególnie w porównaniu z arctan który dąży do asymptoty poziomej i rośnie w okolicach zera dużo szybciej niż sin. Możliwe, że jest to wynik którego nie da się uogólnić na wiele uruchomień tego testu. Zgodnie z oczekiwaniami, test używający funkcji nagradzającej dużą wysokość drzewa osiągnął wysokość najwyższą z całej grupy. Funkcję tego typu można zastosować aby zmusić algorytm do patrzenia na większą ilość argumentów.

Jedynie funkcja 3 dla pary CAD/EUR wygenerowała dostateczne rozwiązanie.

8.4 EnsembleModifier

W przypadku tego modyfikatora mamy 3 regulowane parametry, które mogą promować różne typy osobników. Przy użyciu parametru negativeContribution, wprowadzamy mnożnik bazowej funkcji celu, który używany jest w przypadku gdy wyłączenie oceny danego osobnika powoduje pogorszenie oceny wykonanej przez głosowanie całej populacji.

PositiveContribution odwrotnie do negativeContribution.

Parametr ńoChange"wykorzystywany jest do zmiany funkcji celu w przypadku gdy wykluczenie osobnika nie zmieni klasyfikacji dokonanej przez populację.

Osobniki wpływające na zmianę klasyfikacji całej populacji, mogą być postrzegane jako bardzo wartościowe, gdyż zawierają materiał który albo jest bardzo zły - należy usunąć go z puli, albo bardzo dobry więc lepiej zapewnić przeżycie osobnikowi.

Innym interesującym aspektem tych testów jest to, że ostateczna wartość funkcji celu jest w wynikiem najlepszego osobnika a nie całej populacji. Więc pomimo zorientowania populacyjnego funkcji celu, generujemy jednego najlepszego osobnika.

Wysokości generowanych drzew sięgają od 2 przy ustawieniu

negativeContribution=2.0

noChange=1.0

positiveContribution=0.5

do 9

negativeContribution=1.1

noChange=0.9

positiveContribution=0.8

Oba ustawienia są kontrowersyjne, gdyż karzą osobniki które zwiększały liczbę poprawnie zakwalifikowanych (przez populację) przykładów. Pierwszy na parze CAD/EUR uzyskał dostateczny wynik 0.22.

8.5 Podsumowanie modyfikatorów funkcji celu

Modyfikatory funkcji celu typu FunctionModifier, mają bardziej zrozumiały wpływ na generowane drzewa niż modyfikator Ensemble. Ten drugi uzyskuje powtarzalne dostateczne wyniki na parze CAD/EUR.

8.6 Połączenie modyfikatorów mutacji i funkcji celu

Testy wykonywane były na zbiorach treningowych z cenami od 01/01/2009 do 30/08/2009 oraz zbiorach testowych z cenami 01/09/2010 do 01/01/2010;

Klasyfikacja odbywała się na podstawie 30 poprzednich cen. Pozycja mogła być trzymana przez następne 10 dni.

Modyfikatory wybrane zostały aby spełniały jeden z dwóch warunków:

- modyfikatory równoważą swoje działanie: Zwiększając wysokość mutacji, równocześnie karzemy wysokie drzewa;
- modyfikatory działają synergistyczne: Zwiększamy wysokość mutacji i nagradzamy wysokie drzewa;

Użycie większego zbioru testowego w przypadku instrumentów giełdowych, nie musi być dobrym rozwiązaniem. Algorytm dostosowuje się do trendów które mogą już nie występować. Nie uzyskana została żadna dostateczna wartość funkcji celu na zbiorze testowym. Również sprawność na zbiorze treningowym mocno zmalała.

9 Inne zrealizowane pomysły

Wprowadzona została możliwość zmieniania sposobu podejmowania decyzji w węzłach drzewa. Chcieliśmy mieć możliwość podejmowania decyzji na podstawie dowolnej funkcji, a nie tylko wartości jednego argumentu. Zaimplementowane miały być funkcje statystyczne (jedna istnieje - wartość średnia). Niestety pomysł powstał dość późno, bardzo spowalniał uruchomienie programu i nie było czasu na optymalizację. Zastosowanie takiego rozwiązanie mogłoby pozwolić na:

- tworzenie drzew decyzyjnych łączących różne modele rynków finansowych
- tworzenie drzew decyzyjnych o mniejszych rozmiarach (więcej informacji zawartych w danym węźle)

Drugi pomysł został nam podsunięty przez przykład występujący w bibliotece WATCHMAKER. Jest to programowanie genetyczne. W kodzie źródłowym znajduje się modyfikacja przykładu z biblioteki w celu regresji na testowym zbiorze cen. Dodaliśmy obsługę większej ilości operatorów:

$$exp(x) \tag{6}$$

$$sqrt(x) \tag{7}$$

$$sqr(x) \tag{8}$$

do już istniejących:

$$+ \tag{9}$$

$$- \tag{10}$$

$$> \tag{11}$$

$$ifthenelse \tag{12}$$

Dwie klasy GeneticProrgammingExample i Markets posiadają funkcje "main"i można spróbować je uruchomić. Z wstępnych testów uzyskiwaliśmy błąd ok. 20 punktów. Algorytm potrzebował też bardzo dużo czasu na działanie.

10 Podsumowanie

Niestety nie jesteśmy zadowoleni z wyników algorytmu jeżeli chodzi o klasyfikację rynków giełdowych. Jedyna para na której powtarzalnie uzyskiwane były dostateczne wyniki to CAD/EUR. Ponieważ korelacja tej pary jest słaba, warto byłoby przetestować inne słabo skorelowane pary.

Zaobserwowaliśmy duży wpływ modyfikatorów funkcji celu i mutacji na generowane drzewa, a tym samym na ukierunkowanie algorytmu. Następnym pomysłem byłoby modyfikowanie ukierunkowania w mądrzejszy sposób. Na przykład, wykrywając stagnację - brak poprawy przez kilka generacji, bądź podobną sprawność wszystkich osobników w populacji - zmienilibyśmy algorytm tak aby wprowadzał nowy materiał genetyczny.

Możliwa jest tez, w podejściu hybrydowym, sytuacja w której tak dobieramy algorytm ewolucyjny aby przeciwdziałał wbudowanym ukierunkowaniom drugiej metody.

Podsumowując: zmienianie ukierunkowania jest jednym ze sposobów wpływania na wygenerowane rozwiązanie, ale dzięki naszemu algorytmowi nie staniemy się miliarderami!

Literatura

[1] D. E. Goldberg: Algorytmy genetyczne i ich zastosowania. Warszawa: WNT, 1998.

[2] Paweł Cichosz: Systemy uczące się, WNT, Warszawa 2000

[3] Arabas J., Wykłady z algorytmów ewolucyjnych, WNT, Warszawa 2001

[4] Kwaśnicka H.: Obliczenia ewolucyjne w sztucznej inteligencji, Oficyna Wydawnicza PWr., Wrocław, 1999

[5] Michalewicz Z.: Algorytmy genetyczne + struktury danych = programy ewolucyjne, WNT, Warszawa, 1996

[6] blog: http://tr8dr.wordpress.com/

[7] blog: http://tr8dr.wordpress.com/2009/12/28/10-years/

[8] blog: http://www.puppetmastertrading.com/blog/

[9] blog: http://www.maxdama.com/

[10] Carol Alexander: Market Models: A Guide to Financial Data Analysis, Wiley, 2001

[11] http://www.cs.waikato.ac.nz/ml/weka/

[12] http://watchmaker.uncommons.org/

[13] http://finance.yahoo.com/

[14] http://www.oanda.com/currency/historical-rates