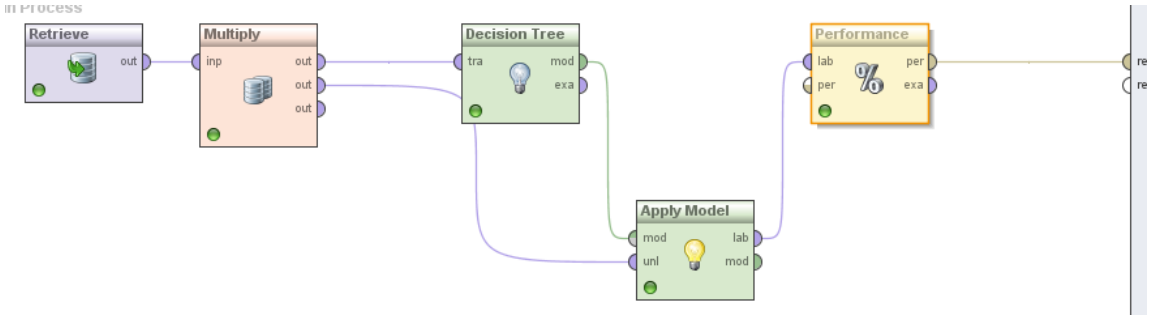
Testowanie modeli predykcyjnych

Wstęp

Podczas budowy modelu, którego celem jest przewidywanie pewnych wartości na podstawie zbioru danych uczących poważnym problemem jest ocena jakości uczenia i zdolności poprawnego przewidywania.

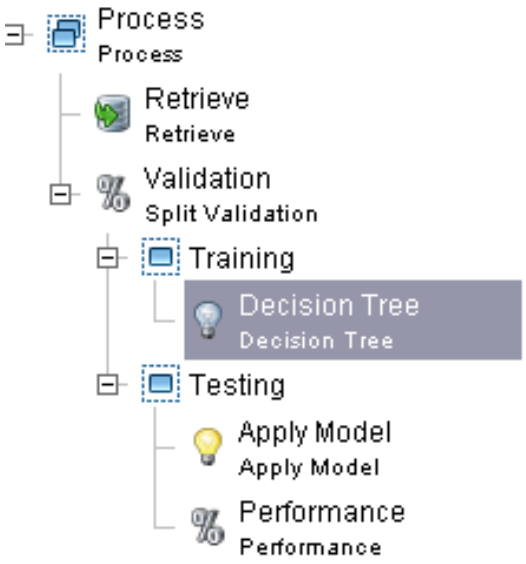
Częstym błędem osób początkujących w zakresie analizy danych jest przeprowadzanie testów na tym samym zbiorze na którym system był uczony:



Takie rozwiązanie nie jest poprawnym miernikiem jakości nauczonego modelu i prowadzi do wyników które są przeszacowane, czyli nadmiernie optymistyczne.

Ponieważ zwykle głównym celem budowy modeli predykcyjnych jest późniejsze wykorzystanie przewidywań modelu na danych niedostępnych podczas procesu uczenia więc opracowano szereg metod pozwalających na znacznie bardziej uczciwy pomiar dokładności.

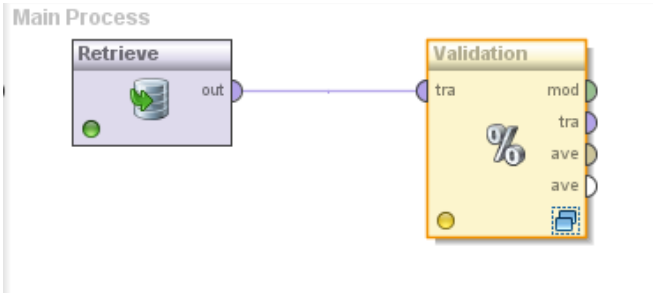
W większości książek poświęconych tematyce sztucznych sieci neuronowych najczęściej spotykanym rozwiązaniem jest podział zbioru dany na dwa niezależne podzbiory tzw. uczący i testowy. Taki podział zwykle realizowany jest w stosunku 70% przypadków to część ucząca zbioru, oraz 30% stanowiąca część testową. Idea oceny modelu lub doboru odpowiednich parametrów modelu sprowadza się wówczas do nauczenia modelu na części uczącej oraz przetestowania go na części testowej, która nie była wykorzystywana w procesie uczenia modelu. Dzieki wydzieleniu dwóch niezależnych podzbiorów wektory części testowej zawierają informację o faktycznym wyniku jaki powinien zostać osiągnięty, natomiast nauczony (na części uczącej zbioru) model dostarcza wyników przewidywań. Taka para pozwala na wyznaczenie odpowiednich wartości oceny typu błąd średniokwadratowy dla modeli regresyjnych lub dokładność klasyfikacji dla modeli klasyfikacyjnych.



W środowisku RapidMiner sprowadza się to do wykorzystania operatora *Evaluation/Validation/SplitValidation*. Operator **SplitValidation** sam dokonuje podziału zbioru danych treningowych na dwa podzbiory (losowanie jest bez zwracania) a następnie w wykorzystuje część treningową zbioru do nauczenia modelu. Nauczony model w kolejnym kroku przekazywany jest do części w której następuje testowanie. W części tej na wejście modelu przekazywane są dane testowe dla których dokonywana jest predykcja wartości. Ostatni krok to policzenie dokładności i zwrócenie jej na wyjście całego systemu.

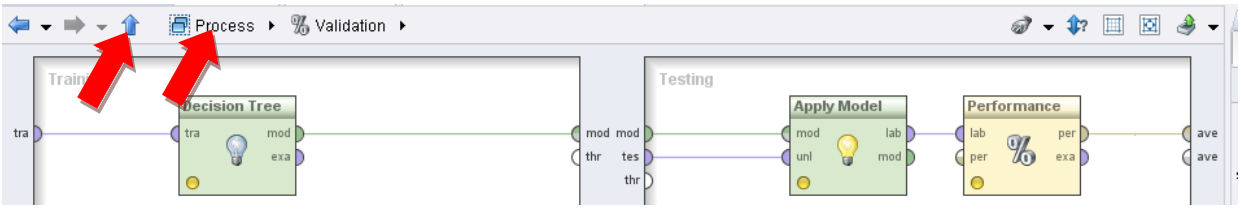
Można to zrealizować zgodnie z poniższym schematem:

Wstaw operator wczytujący zbiór danych zawierający etykiety (jedną ze zmiennych zdefiniowaną jako **label**), a następnie dodaj operator: *Evaluation/Validation/SplitValidation* Na jego wejście podłącz wczytany zbiór danych, jak przedstawia to poniższy rysunek:



Wyjścia operatora *Validation* to: **mod** – model uzywany do uczenia, nauczony na całym zbiorze danych, **tra** – zbiór treningowy (całość – przepisanie wejścia na wyjście), **ave** – średnia dokładność uzyskanych wyników. Wyjście **ave** zwykle podłączamy jako wyjście całości systemu – tak aby wyświetliło się w zakładce *Results.*

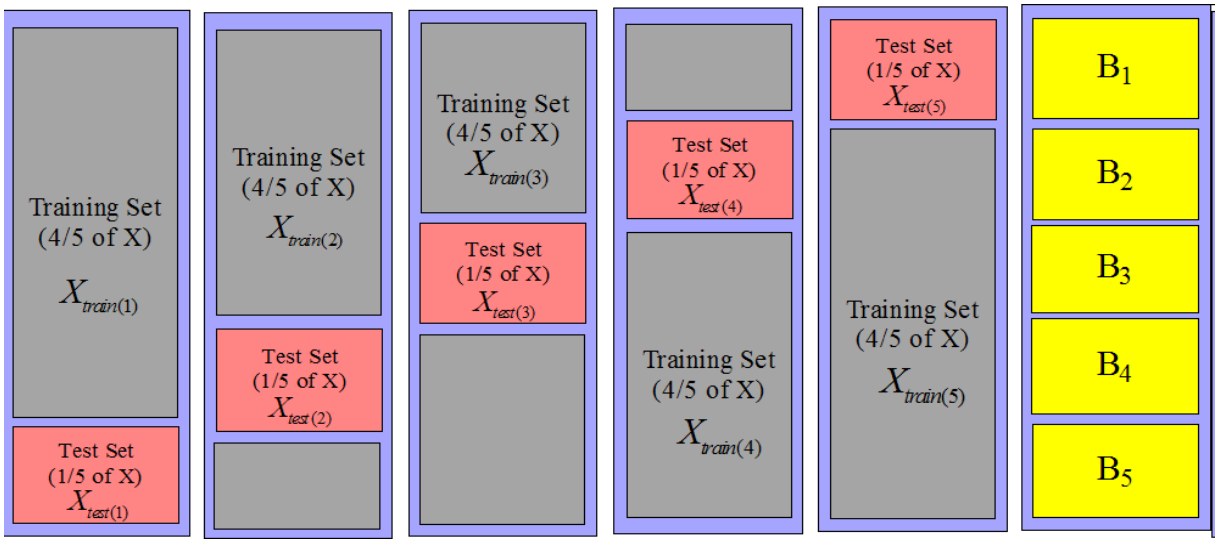
W następnym kroku, aby zdefiniować proces uczenia oraz testowania należy dwukrotnie kliknąć w ikonę *Validation,* co spowoduje otwarcie procesu walidacji. Wówczas wewnątrz tworzymy proces treningu i testowania, jak na przykładowym rysunku:



Wyjść z powyższego okna można klikając w miejsca oznaczone czerwoną strzałką.

Na powyższym rysunku wyjście **tra** oznacza port dostarczający zbioru uczącego. W sekcji *Treningowej* odpowiednio wyjście **mod** oznacza miejsce dostarczenia nauczonego modelu. W części testowej widnieją dwa główne porty wejściowe **mod** oraz **tes.** Pierwsze zawiera nauczony model (istnieje bezpośredni link pomiędzy obydwoma wejściami/wyjściami typu **mod**), natomiast wejście **tes** to zbiór danych testowych. Połączenia w części testowej realizujemy tak samo jak w przypadku nie wykorzystywania testu walidacyjnego. Wyznaczoną dokładność podajemy na wyjście **ave**.

Alternatywą dla podziału trening/test jest powtórzenie tego podziału wielokrotnie, jednak zawsze tak aby część testowa nigdy się nie powtórzyła. Celem tego testu (tzw. testu krzyżowego lub cross-Validation) jest redukcja wpływu losowości na uzyskane wyniki pomiarów. Podział na część treningową i testową przedstawia poniższy rysunek:



gdzie cały zbiór danych został podzielony na 5 równych części (oczywiście części są równe w ramach możliwości). Następnie proces uczenia i testowania modelu uruchamiany jest tyle razy ile różnych części wydzielono (tutaj 5 razy), a następnie dokładność obliczana jest jako średnie z pięciu różnych testów. Liczbę na jaką należy zbiór danych podzielić określa się w opcjach operatora testu krzyżowego.

Aby zbudować wspomniany test należy powtórzyć operacje z poprzedniego przykładu z tą różnicą że tym razem wykorzystujemy operator *Evaluation/Validation/X-Validation.*

Ostatnim z najczęściej stosowanych testów jest tzw. test bootstrap. Jego sposób działania zbliżony jest do testu krzyżowego, jednak z tą różnicą, iż tym razem proces losowania odbywa się ze zwracaniem, co powoduje że kilkukrotnie może zostać wylosowany ten sam wektor. Metoda ta została opracowana przez dwóch naukowców Efron'a oraz Tibshiraniego. Jej głównym zastosowaniem jest analiza przypadków dla małych zbiorów danych, gdzie wielokrotne powtórzenie

tego samego wektora uczącego pozwala na sztuczne zwiększenie rozmiaru zbioru. Wyznaczenie odpowiedniej wartości wyjściowej odbywa się wg. Schematu:

Mając zbiór *X* o *M* wektorach

- Ze zbioru danych *X* wylosuj ze zwracaniem *M* wektorów aby powstał zbiór R_j
- Naucz model H_j na zbiorze R_j i wyznacz błąd modelu na tym zbiorze RE_j
- Wyznacz błąd XE_j modelu H_j na całym zbiorze danych *X*
- Wyznacz optymizm $opt_j=XE_j-Re_j$
- Powtórz kroki 1:4 *J* krotnie
- Wyznacz średni optymizm $\underline{opt}$
- Wyznacz błąd na zbiorze uczącym *E* (trening modelu i jego testowanie odbywa się na zbiorze *X*)
- Na tej podstawie estymatę błędu generalizacji (*e) wynosi*
  $e = E + \underline{opt}$

Operatorem testowania realizującym opisany algorytm jest *Evaluation/Validation/Bootstrapping Validation*

Ostatnim z typowych testów pozwalającym na oszacowanie dokładności jest test typu „Jeden-Pozostaw". Test ten jest szczególnym przypadkiem testu krzyżowego, gdzie zbiór treningowy o *n* wektorach dzielony jest na *n* części uczących, tak aby zawsze jeden i tylko jeden wektor stanowił zbiór testowy.

Zadania

Zadaniem do realizacji jest ocena różnych testów spośród opisanych powyżej, tak aby wyznaczyć metodę testowania dokładności modelu charakteryzującą się największą poprawnością predykcji. W tym celu wczytaj zbiór danych SimpleTrening, a następnie stosując jako model uczący operator *Naivnego-Bayesa* (możesz go znaleźć w *Modeling /Classification and Regression/Bayesian-Modeling/Naive-Bayes*) dokonaj weryfikacji jakości uczenia starając się oszacować średni błąd klasyfikatora Naivnego Bayesa.

1) Zweryfikuj dokładność uczenia klasyfikatora Naivnego Bayesa stosując metodę podziału część ucząca/część testowa, zweryfikuj dokładność dla różnych wartości parametru **Split ratio**
2) Zweryfikuj dokładność uczenia klasyfikatora Naivnego Bayesa stosując metodę testu krzyżowego, zweryfikuj dokładność dla różnych wartości parametru **number of validations**
3) Zweryfikuj dokładność uczenia klasyfikatora Naivnego Bayesa stosując metodę testu typu bootstrap, zweryfikuj dokładność dla różnych wartości parametru **number of validations**
4) Zweryfikuj dokładność uczenia klasyfikatora Naivnego Bayesa stosując metodę testu jeden pozostaw, test ten można wykonać stosując operator *Evaluation/Validation/X-Validation* z zaznaczoną opcją **leave one out**

5) Dokonaj oszacowania prawdziwej jakości uczenia modelu Naivnego Bayesa poprzez nauczenie go na całym zbiorze SimpleTrain i przetestowaniu na zbiorze SimpleTest. Uzyskaną dokładność można uznać za prawdziwą (rzeczywistą) wartość dokładności jaką można uzyskać stosując dany model (tutaj Naivnego Bayesa)
6) Powtórz obliczenia dla różnych parametrów **random seed** i policz średnią z uzyskanych wyników (w celu ułatwienia pracy wykorzystaj operator *loop-parameters(Process Control/Loop/Loop Parameters)* oraz *log (Utility/Logging/log)*)