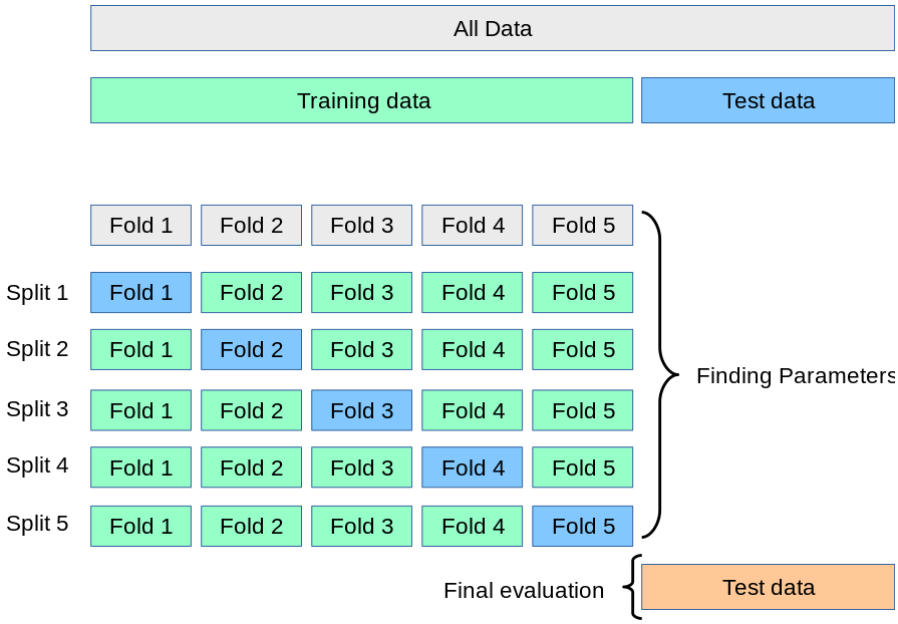
Omówmy od początku, jak działa walidacja krzyżowa (cross-validation, CV). Obrazek z dokumentacji scikit-learn:



Celem jest otrzymanie lepszego oszacowania błędu bez dodatkowych danych. Stosując pojedynczy podział, np. train-valid, mamy dość mały zbiór walidacyjny, więc niezbyt precyzyjne oszacowanie, jak model sobie radzi. Możemy więc przeszacować np. jakość danego zestawu hiperparametrów, albo jej niedoszacować. Aby uzyskać dokładniejsze oszacowanie, stosujemy walidację skrośną. Dla jasności można ją stosować albo do walidacji i doboru hiperparametrów (tniemy zbiór treningowy, foldy po kolei służą za zbiór walidacyjny), lub też do testowania (używamy całego zbioru, foldy są zbiorami testowymi).

Tniemy zbiór na *k* kawałków, np. 5 (5-fold CV). Iterujemy po nich po kolei. W danej iteracji *i*-ty fold jest zbiorem walidacyjnym, a pozostałe łączymy w zbiór treningowy. Trenujemy model na zbiorze treningowym, a potem sprawdzamy zadaną metrykę jakości (np. MSE czy MAE) na zbiorze walidacyjnym. Wynik zapisujemy i idziemy dalej. Dla każdego foldu używamy tych samych hiperparametrów, ale model jest trenowany i sprawdzany na nieco innych danych. Kiedy przejdziemy przez wszystkie *k* foldów, mamy *k* wyników walidacyjnych, a po drodze wytrenowaliśmy *k* modeli. Wyniki uśredniamy i ta średnia stanowi oszacowanie jakości modelu, wynik walidacyjny. Uśrednianie wielu liczb jest dokładniejsze (ma większą moc statystyczną), więc taka miara powinna lepiej mówić, jak dobrze radzi sobie np. dany zestaw hiperparametrów. Dzieje się to zwiększonym kosztem obliczeniowym, bo musimy wytrenować *k* modeli.

Jeżeli używamy CV do tuningu hiperparametrów, to procedurę taką powtarzamy po kolei dla każdego zestawu hiperparametrów (grid search CV). Czyli najpierw definiujemy zakresy wartości hiperparametrów do sprawdzenia, potem budujemy siatkę wszystkich możliwych kombinacji tych własności. Następnie idziemy po kolei, dla każdego zestawu uruchamiamy *k*-fold CV, otrzymujemy z niego uśredniony wynik walidacyjny, zapisujemy taką parę (hiperparametry,wynik). Na koniec wybieramy zestaw hiperparametrów, który osiągnął najlepszy wynik.

Późniejszy model, którego jakość sprawdzamy na zbiorze testowym, jest zazwyczaj retrenowany na całości danych treningowych, bez wycinania zbioru walidacyjnego, używając znalezionych optymalnych hiperparametrów.

Pod kątem praktycznym w scikit-learn implementujemy to jako:

- zakresy wartości hiperparametrów - listy wartości
- siatka hiperparametrów - słownik {"nazwa_hiperparametru": [lista wartości]}
- model - tworzymy instancję
- *GridSearchCV* - realizuje samą walidację skrośną przez przeszukanie wszystkich kombinacji (całej siatki) hiperparametrów

GridSearchCV dostaje jako parametry:

- siatkę hiperparametrów do sprawdzenia
- model jako "czystą", niewytrenowaną instancję
- *scoring* - metryka do optymalizacji, albo string (tabelka dopuszczalnych wartości), albo funkcja o określonych parametrach (dokumentacja)
- *cv* - liczba foldów, lub bardziej złożona strategia splitowania (dokumentacja)

scoring to zwykła funkcja, np. MSE czy MAE. Dla uproszczenia implementacji scikit-learn przyjmuje konwencję, że wewnątrz *GridSearchCV* metryka ma być zawsze maksymalizowana. Dlatego kiedy chcemy dostać model o najlepszym MAE, podajemy (dość nieintuicyjnie) *scoring="neg_mean_absolute_error"*. Średnią wartość tej metryki dla foldów walidacyjnych, dla znalezionego najlepszego zestawu hiperparametrów, można odczytać z parametru *best_score_* obiektu *GridSearchCV*. Predykcje i wartość metryki na zbiorze testowym trzeba już oczywiście policzyć samemu osobno.

GridSearchCV to bardzo ogólny mechanizm - działa zawsze, ale nie wykorzystuje żadnych optymalizacji specyficznych dla algorytmu. Na zajęciach omawialiśmy takie optymalizacje dla ridge regression (SVD + LOOCV) oraz dla LASSO regression (regularization path). Implementują to osobne klasy: *RidgeCV* oraz *LassoCV*.

RidgeCV wykorzystuje całkowicie osobną implementację ze względu na specyfikę tej optymalizacji. Domyślne wartości *scoring=None* oraz *cv=None* skutkują użyciem LOOCV (Leave-One-Out Cross-Validation) oraz MSE jako metryką. Możemy przekazać inną metrykę jako *scoring* bez wpływu na koszt obliczeniowy. Natomiast jeżeli chcemy przyspieszonej implementacji z użyciem SVD, to da się to zrealizować tylko z pomocą domyślnego LOOCV.

LassoCV w każdym kroku używa zwyczajnego *k*-fold CV. Jedyna różnica polega na tym, że sprawdza jedyny hiperparametr (siła regularyzacji) w konkretnej kolejności oraz wykorzystuje wyniki poprzedniej iteracji (słabszej siły regularyzacji). Zwyczajny *GridSearchCV* zakłada, że hiperparametry są od siebie całkowicie niezależne, dlatego musi to być osobna klasa. *LassoCV* zawsze używa tej samej miary jakości, tj. loss = MSE + alpha * L1 (alpha - siła regularyzacji), i wybiera model o najmniejszym koszcie. Nie możemy wybrać innej metryki, bo wtedy regularization path by nie działał. On może działać dlatego, że MSE ma minimum cały czas w

tym samym punkcie (w końcu zbiór danych się nie zmienia), a to alpha rośnie (sprawdzamy coraz większą siłę regularyzacji). Przez to minimum takiej sumy przesuwa się w kolejnych iteracjach (dla większych alpha) w stronę mniejszych wag.