**Validation\_curve()**

* When tuning hyperparameters, we need to keep track of train and test scores and see the behavior of these curves w.r.t various parameters.
* We use GridSearch method for that. But in some cases **one might want to see the effect of a single hyperparameter** and plot it on the graph to see the train and validation scores.
* We can use **validation\_curve** for that.

Args:

1. Model- ex: RandomForestRegressor(n\_estimators=100)
2. X
3. target
4. str: name of parameter to be checked- ex: max\_depth
5. list like: range of values of parameter to be checked
6. int: cv-cross validation folds
7. scoring: An str for metric- to see list of metrics use *sorted(sklearn.metrics.SCORERS.keys()) ------ (I)*

Returns:

1. train scores: list
2. validation scores: list

**sklearn.metrics.make\_scorer()**

* So a lot methods like GridSearchCV, validation\_curve, cross\_val\_score etc.. mostly related to validation will accept something called as a **scorer.**
* There are multiple ways in which you can score models in sklearn. In some functions u need to input scoring parameter. In others there are default scorers as a part of the model API itself. Thirdly there are also individual metric functions in sklearn.metrics.
* When should one use what will only be clear with practice but most **validation** functions take scoring parameter/scorers. And in that parameter either u can give an str value found in *(I)* above, or you can convert a metric function (either custom or from sklearn.metrics) into a **scorer** object using a factory function provided by sklearn.metrics called as **make\_Scorer()**

*From sklearn.metrics import mean\_squared\_error, make\_scorer*

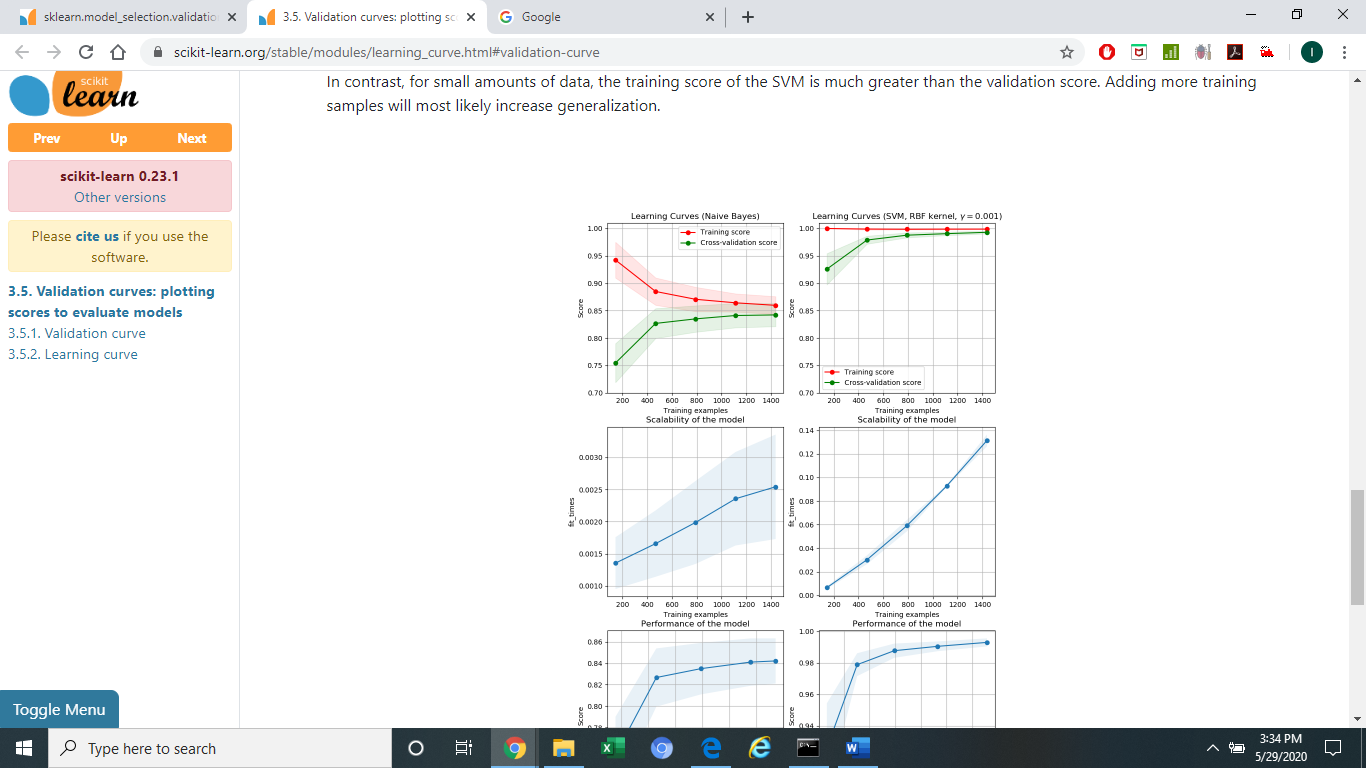
*Mse\_scorer = make\_scorer(mean\_squared\_error)*

**Difference between neg\_\*\*regression\_metric\*\* and \*\*regression\_metric\*\* example: neg\_mean\_squared\_error and mean\_squared\_error is that these two are just the value flips of each other.**

* In *(I)* above you will notice that ‘mean\_squared\_Error’ does not exist instead for regression metrics *neg* keyword is prefixed.
* This is just for design purposes of scorers and how they fit into sklearn.
* So *neg\_mean\_squared\_error* is just *mean\_squared\_error* but it is exactly flipped.

**Learning curves**

* A type of a process which results validation and train scores in which we again plot train and validation scores for a model w.r.t varying training sample numbers.
* Ex: refer to the following plot



* The left plot has *learning\_curve* plotted for Naïve bayes classifier and the right one is for SVM classifier. The score is accuracy.
* We can see that as training example increase both train and cv score converge in NB classifier. This indicates increasing data would not help as they have almost converged and validation score would not get better.
* For SVC though the training score has never come down. Ideally as we increase our train data, the train score comes down so definitely increasing the data would help in generalizing the model.
* The exact use of this plot is not yet clear to me but for now I am going with only 1 important point **“If you ever end up asking yourself whether I should collect more data? Or if you end up asking how much data is enough to get the best result for the current model? – You should turn to LEARNING CURVES”**.

Args:

1. Model- ex: RandomForestRegressor(n\_estimators=100)
2. X
3. target
4. list like: range of proportions between 0 and 1 for the training samples
5. int: cv-cross validation folds
6. scoring: An str for metric- to see list of metrics use *sorted(sklearn.metrics.SCORERS.keys()) ------ (I)*

**GridsearchCV**

* Visualizing train-validation scores for 1 parameter is limiting. We usually have multiple hyperparameters for our models.
* Plots in this case become complex as there are more than 2 dimensions.
* So in such cases we usually only want to get the parameter values that performed the best on a particular dataset.

*From sklearn.model\_validation import GridSearchCV*

*From sklearn.ensemble import RandomForestRegressor*

*rf\_param\_grid = {'max\_depth': [3, 5, 7, 9],*

*'min\_samples\_split': [50, 100, 150],*

*'max\_features': [0.3, 0.5, 0.7]*

*}*

*rf\_grid = GridSearchCV(RandomForestRegressor(n\_estimators=100), rf\_param\_grid, cv=7)*

*rf\_grid.fit(X, y)*

*# training start time - 17:52*

*rf\_grid.best\_params\_*

* To get detailed args please refer to sklearn docs.