Project: scikit-learn

2013-01-16T07:09:01Z jwkvam 73comments

Fitting additional estimators for ensemble methods

Web url: https://github.com/scikit-learn/scikit-learn/issues/1585

API url: https://api.github.com/repos/scikit-learn/scikit-learn/issues/1585

I would like to propose an additional instance method to the ensemble estimators to fit additional sub-estimators. I kluged up an implementation for gradient boosting that appears to work through my limited testing. I was thinking the signature would be something like

``` python

def fit\_extend(self, X, y, n\_estimators):

```

where `self.n\_estimators += n\_estimators` is updated as so. I don't think fit\_extend is a particularly great name so I'd welcome other suggestions. Perhaps we would want to hash the features and labels when fit() is called so we can check that the same features and labels are provided to this function.

If people think this would be a useful addition I would be willing to put together a PR, it seems like it should be straightforward to implement and add tests/docs for.

-------------------------------------------------------------------------

2013-01-23T23:04:52Z amueller

This is definitely a feature we want. The question is: what would be the best way to implement it (in terms of API)?

There is something slightly similar in the adaboost pr: #522. That implements predicting with a subset of the estimators, which is also very helpful.

What do you think does the scenario / code look like, where a user wants `fit\_extend`? It is probably most useful in an interactive setting, righ?

There is a slightly related function in SGD, `partial\_fit`. That is actually for online learning, though, so it gets different data.

I'd like to get this feature with adding as little API an names as possible ;)

-------------------------------------------------------------------------

2013-01-23T23:05:55Z amueller

Btw, I wouldn't hash `X` and `y` . I don't see a reason to force the user to provide the same input data.

-------------------------------------------------------------------------

2013-01-25T06:02:17Z jwkvam

I would like to train a small number of sub-estimators at a time (and wait a relatively short time). Then test it on my cross-validation set and if my cv score is still falling, I can continue training. As opposed to training a large number of sub-estimators and waiting a long time (several hours for me). That was my motivation.

I can understand being hesitant about adding another instance method. I thought it might be worthwhile to add another optional parameter to fit() but I saw this quote on the contributing page.

> fit parameters should be restricted to directly data dependent variables

So I wasn't sure that would be a good idea. Would

``` python

def fit(self, X, y, n\_estimators=self.n\_estimators)

```

be acceptable? Then if `n\_estimators > self.n\_estimators`, we'll then train that many more estimators.

I agree that adding in n\_estimators parameter to the prediction method is nice, but I think you'll agree that it solves a different problem. For my problem performing grid search over n\_estimators isn't really an option because it takes so long.

-------------------------------------------------------------------------

2013-01-25T07:36:59Z glouppe

Until we agree on a proper interface to do that, you could use the following hack:

```

# Train a forest of 10 trees

clf1 = RandomForestClassifier(n\_estimators=10)

clf1.fit(X, y)

# Train a second forest of 10 trees

clf2 = RandomForestClassifier(n\_estimators=10)

clf2.fit(X, y)

# Extend clf1 with clf2

clf1.estimators\_.extend(clf2.estimators\_)

clf1.n\_estimators += clf2.n\_estimators

# clf1 now counts 20 trees

```

-------------------------------------------------------------------------

2013-01-25T07:38:11Z glouppe

Note that this only work for RandomForest and ExtraTrees. The same trick cannot be used with Gradient Boosting.

-------------------------------------------------------------------------

2013-01-26T17:14:06Z amueller

See #1626. Would early stopping be an acceptable solution to you?

-------------------------------------------------------------------------

2013-01-29T07:20:22Z jwkvam

@amueller I share the same opinion as @glouppe here https://github.com/scikit-learn/scikit-learn/issues/1626#issuecomment-12785168. I like early stopping but it doesn't resolve this in my opinion.

-------------------------------------------------------------------------

2013-01-29T08:29:59Z amueller

Ok. Then we should look for a solution that allows for early stopping and adding additional estimators.

-------------------------------------------------------------------------

2013-01-29T09:45:19Z amueller

Thinking about it a bit more, I think the `partial\_fit` method would be the right interface. In SGD you can call `partial\_fit` either with the same data or new data and it keeps on learning. The difference is that in SGD, if you manually iterate over batches, you get the original algorithm out. For ensembles, that would not be true. You would need to use the whole data on each call to `partial\_fit`.

-------------------------------------------------------------------------

2013-01-29T15:49:58Z GaelVaroquaux

> Thinking about it a bit more, I think the partial\_fit method would be the right

> interface.

I like this suggestion. What do other people think?

-------------------------------------------------------------------------

2013-01-30T07:26:41Z glouppe

Just to clarify, what would exactly happen in `partial\_fit` in case of ensembles? Would that add `n\_estimators` more estimators, where`n\_estimators` is the parameter value from the constructor? (or could we change that value?)

-------------------------------------------------------------------------

2013-01-30T08:34:18Z amueller

Good question. I also thought about that ;) actually, you would want to change that, right? you could change that afterwards by `set\_params` but that feels awkward :-/

-------------------------------------------------------------------------

2013-01-30T09:34:06Z pprett

sorry for joining the discussion so late.

I agree that we need such a functionality, however, I'm not sure if `fit\_extends` is the best solution to the problem that @jwkvam describes. In order to do early stopping the user has to write some some code that basically repeatedly calls `fit\_extends` and then checks the CV error.

I'd rather propose the `monitor` fit parameter that we discussed in the past: `est.fit(X, y, monitor=some\_callable)` where `some\_callable` will be called after each iteration and is passed the complete state of the estimator. The callable could also return a value whether or not the training should proceed.

Using such an api one could implement not only early stopping but also custom reporting (e.g. interactive plotting the training vs. testing score) and snapshoting (all X iterations dump the estimator object and copy it to some location; this is great if you are running on EC2 spot instances or some other unreliable hardware ;-)

Even with such a `monitor` API, however, I think there would be a need for an API to fit more estimators once the model has been fitted (i.e. `fit\_extends`) - often one trains a model and does some introspection to find that its probably better to have run more iterations - existing estimators use the `warm\_start` parameter to implement such a functionality (e.g. see `linear\_model.ElasticNet`) - here is the docstring of the parameter::

```

warm\_start : bool, optional

When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.

```

Personally, I'd prefer `fit\_extends` (or `fit\_more`) over `warm\_start` - warm start is quite implicit - you have to::

```

est = GradientBoostingRegressor(n\_estimators=1000)

est.fit(X, y)

# now we want to fit more estimators to ``est``

# if you forget warm\_start=True you nuke your previous estimators - quite implicit

est.fit(X, y, n\_estimators=2000, warm\_start=True)

# alternatively - more explicit

est.fit\_more(X, y, n\_estimators=1000)

```

-------------------------------------------------------------------------

2013-01-30T10:03:20Z GaelVaroquaux

> # alternatively - more explicit

>

> est.fit\_more(X, y, n\_estimators=1000)

To me, fit\_more corresponds really to the partial\_fit that we have in

other estimators.

-------------------------------------------------------------------------

2013-01-30T10:10:46Z amueller

@pprett I think there should be an easy way to do easy things. a monitor api is very flexible but actually you want to do early stopping \*\*every time\*\* you use an estimator, right? So there should be no need to write a callback to do that. Also, it must be compatible with GridSearchCV.

-------------------------------------------------------------------------

2013-01-30T10:16:24Z ogrisel

> To me, fit\_more corresponds really to the partial\_fit that we have in

> other estimators.

I don't think so. In `partial\_fit`, "partial" stands for partial access to the data: you expect that the data does not fit in memory at once so you fit with one chunk at a time and update the model incrementally while scanning through the data.

In this case we want to change the number of sub estimators but might want to reuse exactly the same data at each call.

For a similar reason ElasticNet has a `warm\_start` constructor param instead of a `partial\_fit` method and SGDClassifier both has a `warm\_start` param and a `partial\_fit` method: they serve different purposes.

I agree that the monitor API would be very useful in general (for dealing with snapshoting, early stopping and such) but would not solve the issue of growing the number of sub-estimators in an interactive manner.

We could also have:

```

est.fit(X, y, n\_additional\_estimators=1, warm\_start=True)

```

Or even to grow by 110% (10% more estimators):

```

est.fit(X, y, additional\_estimators=0.1, warm\_start=True)

```

-------------------------------------------------------------------------

2013-01-30T10:19:36Z amueller

hum I didn't look to much into the warm start api that we have currently. There is no central documentation for that, right?

We should really think about the organization of the docs. We got quite some comments on that in the survey :-/

-------------------------------------------------------------------------

2013-01-30T10:22:38Z amueller

@ogrisel I'd have to have a look at the SGD implementation to see the details but what is the difference in what actually happens between warm-starts and partial\_fit? I think we agree on the point of same /changing data.

Does `warm\_start` do several epochs and `partial\_fit` does not? That would make sense to me, and then we should probably keep them separate.

If we already have the warm-start api, we should definitely "just" implement that for the ensemble estimators.

-------------------------------------------------------------------------

2013-01-30T10:39:38Z ogrisel

`warm\_start` just prevents fit to forget about the previous state (assuming that the inner state of the model will likely make it converge faster to the solution of the new call with the new hyperparameter).

-------------------------------------------------------------------------

2013-01-30T10:41:59Z pprett

2013/1/30 Andreas Mueller notifications@github.com

> @ogrisel https://github.com/ogrisel I'd have to have a look at the SGD

> implementation to see the details but what is the difference in what

> actually happens between warm-starts and partial\_fit? I think we agree on

> the point of same /changing data.

> Does warm\_start do several epochs and partial\_fit does not? That would

> make sense to me, and then we should probably keep them separate.

> If we already have the warm-start api, we should definitely "just"

> implement that for the ensemble estimators.

I think the main difference is the \_semantics\_: the main idea behind

`warm\_start` is to converge more quickly - but no matter what value

`warm\_start` has you get the same solution!

Partial fit on the other hand, changes the underlying model. Consider the

following example:

```

# this is the intended use-case for warm\_start is faster convergence

clf = SGDClassifier(n\_epochs=10)

clf.fit(X, y)

clf2 = clone(clf)

clf3 = SGDClassifier(n\_epochs=10)

clf2.fit(X, y, warm\_start=True)

clf3.fit(X, y)

# clf2 and clf3 should converge to the same solution - but since clf3

```

can reuse the fitted weights from clf it might converge more quickly

# under the hood `SGDClassifier.fit` resets the "training" state mode

the estimator (adaptive learning rate for sgd)

```

# now partial fit

clf = SGDClassifier(n\_epochs=10)

clf.partial\_fit(X, y, classes)

# training has not completed yet "training" state (adaptive learning rate) is stored.

clf.partial\_fit(X, y) # resume with previous learning rate

```

Disclaimer: This example might be pedantic because the differences in terms

of the learned weights is minimal - but conceptually they are IMHO totally

different things...

> ‚Äî

> Reply to this email directly or view it on GitHubhttps://github.com/scikit-learn/scikit-learn/issues/1585#issuecomment-12883146.

##

Peter Prettenhofer

-------------------------------------------------------------------------

2013-01-30T10:47:39Z ogrisel

The `warm\_start` API was initially introduced to allow faster computation of a series of identical linear models when using a path of regularizers `alpha`. This is somewhat similar to iteratively growing the number of sub-estimators in a boosted ensemble model so we could decide to reuse `warm\_start` to adress that use case as well but if this API reveals cumbersome for boosted models it might be better to rethink it now that we have an additional use case.

-------------------------------------------------------------------------

2013-01-30T10:48:38Z ogrisel

I agree with @pprett's analysis.

-------------------------------------------------------------------------

2013-01-30T11:03:58Z amueller

I don't know what to make of @pprett analysis.

In the case of linear models, the estimator will converge to the same result, even when the warm start gets different data than the original fit. If we "warm started" ensembles / trees, that would not be the case.

We could try to assure that the data provided when warm starting is the same as the original.

At the moment, "warm start" refers to an optimization procedure, which there is none in tree based methods.

While `partial fit` retains all of the state of the estimator and just keeps on fitting.

On the other hand, subsequent calls to partial fit on batches lead to the same model as training on the whole data.

Again, this is different from the tree/ensemble case. I feel this goes back to my argument that this is more of a path algorithms than anything else ;)

-------------------------------------------------------------------------

2013-01-30T11:06:47Z amueller

So I see two possible solutions: make sure warm-start is always called with the same data, then adding estimators would be warm starting.

If not, we need a third way to refit a given model.

Where are the docs for that currently, btw ;)

-------------------------------------------------------------------------

2013-01-30T11:36:55Z ogrisel

> make sure warm-start is always called with the same data.

Why so? Let the user decide how and what for he / she want to use `warm\_start` for.

-------------------------------------------------------------------------

2013-01-30T11:40:16Z ogrisel

> Where are the docs for that currently, btw ;)

http://scikit-learn.org/dev/modules/generated/sklearn.linear\_model.ElasticNet.html

```

warm\_start : bool, optional

When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.

```

I agree that giving motivation would be helpful, for instance in this case:

"This is useful to efficiently compute a regularization path of ElasticNet models as done by the :func:`enet\_path` function".

-------------------------------------------------------------------------

2013-01-30T12:18:38Z amueller

I thought the argument was about semantics. I think a semantic is defined by giving the user some guarantee of what will happen. That way the user doesn't need to know all the details of the algorithm.

I thought the guarantee of `warm\_start` was "warm\_start doesn't change the result", while the guarantee of `partial\_fit` was "iterating over batches doesn't change the result".

If there is no guarantee, then I don't see how there can be common semantics.

-------------------------------------------------------------------------

2013-01-30T13:08:48Z ogrisel

We provide guarantee to the user that if he provides the same data again with warm\_start=true he will get the same results (just faster). But we should not prevent the user to use different data if he makes an informed guess that warm starting on the new data will help him solve his problem (e.g. solving on the new data faster if he makes the assumption that the new data is distributed reasonably similarly to the first data and hence starting the optimizer from the previous position should speeds things up).

-------------------------------------------------------------------------

2013-01-30T13:10:26Z amueller

For linear estimators that is ok. But if you want to use `warm\_start` on ensembles, it will have a very different semantic all of a sudden.

-------------------------------------------------------------------------

2013-01-30T13:21:45Z ogrisel

Indeed growing a boosted ensemble on changing data is weird and probably useless (unless if it's a way to inject some randomization for some meta-meta-ensemble estimator that does bagging on boosted models maybe?). I don't think we should try to enforce that the data does not change across calls though. Let's just document the expected usage scenario for that option in the docstring instead.

-------------------------------------------------------------------------

2013-01-30T13:23:33Z amueller

ok. So basically the docstring should say "use warm\_start with the same data unless you know exactly what you are doing".

Fine with me. Anyone opposed to using `warm\_start`?

-------------------------------------------------------------------------

2013-01-30T13:24:26Z amueller

I still have to have a look at how that is handled in SGD and ENet, though...

-------------------------------------------------------------------------

2013-01-30T13:24:59Z GaelVaroquaux

> Indeed growing ensemble on changing data is weird and probably useless

No, not useless: it's one specific sub-sampling strategy. The practical

difference with an online method is that you want the batch to be big.

-------------------------------------------------------------------------

2013-01-31T12:18:22Z amueller

On 01/30/2013 11:42 AM, Peter Prettenhofer wrote:

> I think the main difference is the \_semantics\_: the main idea behind

> `warm\_start` is to converge more quickly - but no matter what value

> `warm\_start` has you get the same solution!

> Thinking about it, this is wrong.

> In SGDClassifier, if you fit twice with `warm\_start` you will definitely

> get different solutions. Depending on what you did before, you might get

> better or worse solutions, but the training time will be exactly the same.

So `warm\_start` in SGDClassifier can not be used for model selection.

On the other hand, `partial\_fit` could be used to find the best

`max\_iter`.

The more I think about it, the more confusing it gets for me :-/

Btw, is there any reason that `warm\_start` is an init parameter

and `partial\_fit` is a function?

Wouldn't it be easier if `partial\_fit` also was an init parameter?

-------------------------------------------------------------------------

2013-01-31T12:23:23Z GaelVaroquaux

> Btw, is there any reason that `warm\_start` is an init parameter

> and `partial\_fit` is a function?

Because `partial\_fit` is a specific strategy that might differ from the

strategy used in `fit`.

> Wouldn't it be easier if `partial\_fit` also was an init parameter?

I think it would be confusing. The goal of `partial\_fit` is to be a

building block usable in an out-of-core framework. Using `fit` for this

purpose could lead to fairly catastrophic results.

-------------------------------------------------------------------------

2013-01-31T12:25:40Z amueller

I don't understand your argument. What `fit` does is basically "forget model, call `partial\_fit`".

-------------------------------------------------------------------------

2013-01-31T12:27:27Z amueller

Hm maybe what you mean is that `fit` might need to do less work than `partial\_fit` because `partial\_fit` needs to store the "sufficient statistics" of the previous data and fit doesn't need to do that?

-------------------------------------------------------------------------

2013-01-31T12:28:15Z GaelVaroquaux

> I don't understand your argument. What fit does is basically "forget model,

> call partial\_fit".

It can do more. Typically it shuffles the data before calling partial

fit. It may also divide it into mini batches of a user-selectable size.

-------------------------------------------------------------------------

2013-01-31T12:29:11Z GaelVaroquaux

> Hm maybe what you mean is that fit might need to do less work than partial\_fit

> because partial\_fit needs to store the "sufficient statistics" of the previous

> data and fit doesn't need to do that?

It might be the case. It might be also that fit needs to do additional

work to turn a large batch dataset into a set of mini-batch ones.

-------------------------------------------------------------------------

2013-01-31T12:45:15Z amueller

hm ok maybe this is not so important right now.

I'd like to minimize the number of mechanisms we have in sklearn, and we definitely need one (more?) for efficient model selection. In the coordinate decent algorithms, the `warm\_start` option was introduced exactly for this purpose. I am not sure it is general enough to really do that (what if there is more than one parameter?) and it doesn't fulfil this requirement any more in SGDClassifier.

-------------------------------------------------------------------------

2013-01-31T12:46:11Z amueller

(just removed a lot of the previous comment as I was repeating myself).

-------------------------------------------------------------------------

2013-01-31T13:27:07Z ogrisel

> I'd like to minimize the number of mechanisms we have in sklearn, and we definitely need one (more?) for efficient model selection. In the coordinate decent algorithms, the warm\_start option was introduced exactly for this purpose. I am not sure it is general enough to really do that (what if there is more than one parameter?) and it doesn't fulfil this requirement any more in SGDClassifier.

I don't understand this last remark. warm\_start is perfectly valid for SGDClassifier (in addition to `partial\_fit`): right now SGDClassifier does not have convergence check / early stopping. But as soon as it has, warm\_starting will make it possible to compute the regularization path faster, exactly as for ElasticNet.

-------------------------------------------------------------------------

2013-01-31T13:48:49Z amueller

SGDClassifier does `n\_iter` epochs of updates, then stops. Where it ends up after `n\_iter` steps depends heavily on where you started.

Even if you do "early stopping", this would be early stopping on the validation set, not early stopping of the optimization. SGDClassfier does not have the goal to fully optimize the objective to the end. So where you will end up will depend on the initialization.

In particular, for early stopping (on a validation set!) it could be better to do less iterations, leading to lower bias.

-------------------------------------------------------------------------

2013-01-31T13:52:58Z amueller

In particular, I don't think a "regularization path for alpha" makes sense in the SGD setting. The "path" is a sequence of optima. SGD will never find the optimum, so the places you'll end up will probably depend as much on the scaling of the learning rate as on the actual regularization.

-------------------------------------------------------------------------

2013-01-31T18:48:43Z ogrisel

> In particular, I don't think a "regularization path for alpha" makes sense in the SGD setting. The "path" is a sequence of optima. SGD will never find the optimum,

For linear models, the problem is convex. If n\_iter is big enough, SGD with a good learning schedule will converge to the optimum (if you don't stop before convergence). The convergence speed when getting closer to the optimum is just not as good as coordinate descent but this is a different issue.

-------------------------------------------------------------------------

2013-01-31T19:59:43Z amueller

so we agree: the models will be different unless n\_iter is big enough and the schedule is just right - which are unlikely in practice.

also a guarantee of the form ‚Äúresults will be the same if the other settings are appropriately tuned‚Äú doesn't really sound like a guarantee.

Olivier Grisel notifications@github.com schrieb:

> > In particular, I don't think a "regularization path for alpha" makes

> > sense in the SGD setting. The "path" is a sequence of optima. SGD will

> > never find the optimum,

>

> For linear models, the problem is convex. If n\_iter is big enough, SGD

> with a good learning schedule will converge to the optimum (if you

> don't stop before convergence). The convergence speed when getting

> closer to the optimum is just not as good as coordinate descent but

> this is a different issue.

>

> ---

>

> Reply to this email directly or view it on GitHub:

> https://github.com/scikit-learn/scikit-learn/issues/1585#issuecomment-12958852

##

Diese Nachricht wurde von meinem Android-Mobiltelefon mit K-9 Mail gesendet.

-------------------------------------------------------------------------

2013-02-01T14:15:16Z amueller

So what about

```

clf = RandomForestClassifier(n\_estimators=10)

clf.fit(X, y)

print(clf.score(X, y))

clf.set\_params(warm\_start=true, n\_estimators=20)

clf.fit(X, y)

```

Is that an acceptable usage pattern?

-------------------------------------------------------------------------

2013-02-01T14:17:06Z amueller

Or do you want these as parameters to `fit`? In SGD, `warm\_start` is an `\_\_init\_\_` parameter according to the docs.

-------------------------------------------------------------------------

2013-02-08T13:19:18Z amueller

Let's revive the discussion. in #1044 @GaelVaroquaux said he still prefers `partial\_fit`.

Currently, I think `warm\_start` is more in the right direction, but I don't have a strong opinion. @ogrisel @pprett @glouppe @larsmans what is your opinion on the usage pattern I posted above? Or would you like to have another interface using `warm\_start` or `partial\_fit`?

-------------------------------------------------------------------------

2013-02-08T14:38:47Z GaelVaroquaux

> Currently, I think warm\_start is more in the right direction, but I don't have

> a strong opinion.

What I dislike about using the 'warm\_start' is that currently the

contract with scikit-learn estimators is that you can call 'fit' and get

a valid/useful answer regardless of the history of the object. It may go

faster or slower, but it's somewhat fool proof. If you pass different

data to an ensemble estimator, and use the 'warm\_start' to fit more

estimators, you will get nonsens. I am worried about having to write

'defensive' code to avoid such problems.

-------------------------------------------------------------------------

2013-02-08T14:55:29Z pprett

how would `partial\_fit` work in our setting - is this correct::

```

est = GradientBoostingRegressor(n\_estimators=1000)

est.fit(X, y)

...

est.fit\_partial(X, y, n\_estimators=1000) # train another 1000

```

so it would take arbitrary `fit\_params` or just `n\_estimators`?

Personally, I'm in favor of a `fit\_more` since the use-case that our current `partial\_fit` serves is quite different and `fit\_more` is more explicit.

-------------------------------------------------------------------------

2013-02-08T17:40:44Z glouppe

I am also not very happy with the name `partial\_fit` in case of ensembles. From my point of view, that name suggests that it will build some estimators out of the total number requested in the constructor, but not more.

If we go for `warm\_start` then what would be the specification? You set `n\_estimators` in the constructor and calling `fit` append `n\_estimators` more estimators? Just like @amueller did above? Well I am not against that pattern, but that does not seem very intuitive to me nevertheless.

From a very practical point of view, I like `fit\_more`. It is explicit. No explanation required. However, it adds another function to our API...

(I have no strong opinion yet, these remarks simply reflect what I think at the moment)

-------------------------------------------------------------------------

2013-02-08T18:06:21Z amueller

I am not completely against adding a function, but I wouldn't like it to be to specific to the ensembles.

I really do see a connection to the path algorithms so I think sharing an interface would be nice.

Consider the following hypothetical situation (maybe not so realistic):

You fitted an ensemble but now you see that you underfit and want to make your trees deeper (let's say we implemented that). This would be another example of path-like behavior. Would you also do that via `fit\_more`? Or add a `fit\_deeper` function?

I guess there is a trade-off between generality and explicitness.

-------------------------------------------------------------------------

2013-02-08T18:09:30Z amueller

@GaelVaroquaux The contract with `partial\_fit` is imho that if you iterate over the data in batches, you will get the same result out. That will definitely not be the case if used here. So by design we would break the contract ?!

-------------------------------------------------------------------------

2013-02-11T21:08:40Z amueller

Thinking about it again, maybe there is room for a new method which we could use to implement #1626.

I wouldn't mind calling it `fit\_more`, but in the sense of `do some more fitting along the parameter path` not in the sense of `fit additional estimators in the ensemble`.

So imho we should either do `warm\_start` ( + maybe defensive programming ) or add another method that we can generally use to fit along a parameter path.

-------------------------------------------------------------------------

2013-02-11T21:14:20Z amueller

Would `fit\_more` then be defensive or not? ;)

-------------------------------------------------------------------------

2013-02-11T22:04:23Z glouppe

-1 on defensive. I'd rather document it well and let the user decides what

is good for oneself.

On 11 February 2013 22:14, Andreas Mueller notifications@github.com wrote:

> Would fit\_more then be defensive or not? ;)

>

> ‚Äî

> Reply to this email directly or view it on GitHubhttps://github.com/scikit-learn/scikit-learn/issues/1585#issuecomment-13403244.

-------------------------------------------------------------------------

2013-02-11T22:06:16Z amueller

I would also be against defensive. I was just wondering if adding the function really solved an issue or if we just added another way to do warm starts. Both have the same defensive / not-defensive problem, right?

-------------------------------------------------------------------------

2013-02-12T04:49:06Z jwkvam

My apologies if I'm simply repeating what has already been said. But it seems like you could split estimators into two classes: those that freeze parameters once they are fit (ensembles, DTs), and those that don't (linear models). By that I mean with warm\_start you won't refit the first n sub-estimators of an ensemble or the existing splits in a decision tree. The lack of being able to reach anywhere in the parameter space with warm\_start for ensembles and DTs makes me think that an instance method would be more appropriate.

If an instance method is chosen, does it need to be more general as @amueller noted? If at some point someone wanted the ability to increase the max\_depth of the sub-estimators, that could also be handled with `fit\_more()`?

For what it's worth, I would also be against defensive. As @GaelVaroquaux pointed out earlier it provides a sub-sampling strategy, for instance, if your training data doesn't fit in main memory.

-------------------------------------------------------------------------

2013-02-13T09:07:54Z glouppe

After some thoughts, I think we should see the bigger picture here. In a near future, I would like to implement generic meta-ensembles that could combine any kind of estimators together. What I rather see is a "combination" mechanism that would take as input a list of (fitted) estimators and would produce a meta-estimator combining them all.

In practice, I think we can achieve that without adding any new function to our API. For example, one could simply pass such a list of fitted estimators to the constructor of the meta-ensemble.

In terms of API, one could (roughly) implement such ensembles in the following way:

a) Bagging:

- constructor: `base\_estimator` (optional), `n\_estimators` (>=0), a list `L` of fitted estimators (optional).

- fit: extend `L` with `n\_estimators` new instances of `base\_estimators` fitted over (bootstrap copies of) the training samples. If no base estimator is given, then it is equivalent to combining the estimators in `L`.

b) Stacking:

- constructor: `base\_estimator` (optional), `n\_estimators` (>=0), a list `L` of fitted estimators (optional).

- fit: extend `L` with `n\_estimators` new instances of `base\_estimators` fitted of bootstrap samples, then refit a model over the predictions of the estimators.

c) Forest:

- constructor: `base\_estimator` (optional), `n\_estimators` (>=0), a list `L` of fitted estimators or a forest (optional).

- fit: extend `L` with `n\_estimators` new instances of `base\_estimators` fitted over the training samples. Here we could also check whether the estimators in `L` are forests or decision trees. Forests would be flattened in order to put all trees on the same level.

Also, in such a framework, computation of an ensemble could easily be distributed over several machines: build your estimators; pickle them; then recombine them into one single meta-estimator. One could even wrap that interface into a MapReduce cluster, without digging into our implementation at all!

What do you think? I am aware this is only relevant to some kind of ensembles though. For instance, GBRT and AdaBoost are (in my opinion) more suited to either `warm\_restart` or `partial\_fit`.

-------------------------------------------------------------------------

2013-02-13T09:19:35Z glouppe

Just to be clear, to extend a forest, one would do something like:

```

forest = RandomForestClassifier(n\_estimators=100)

forest.fit()

forest\_extended = RandomForestClassifier(n\_estimators=100, L=forest)

forest\_extended.fit() # now counts 200 trees

```

-------------------------------------------------------------------------

2013-02-13T10:24:30Z amueller

What is the motivation of that interface? I am totally with you in supporting more ensemble methods. I just feel it is quite awkward to have a different interface for GBRT and random forest. I don't really see the motivation for that.

If the main motivation is to distribute embarassingly parallel jobs, then I think we should attack this by implementing a more powerful parallelization. Doing it the way you described seems pretty manual and hacky.

Basically I feel your proposal just solves a very special case and leaves most cases unsolved.

-------------------------------------------------------------------------

2013-02-13T10:31:30Z glouppe

Well ok... I just feel that extending boosted-like ensembles and average-like ensembles are quite different things.

-------------------------------------------------------------------------

2013-02-13T10:32:30Z amueller

What is the use-case for your interface except parallelization? Or better: in what use cases do you need a different interface for boosted ensembles and bagging?

-------------------------------------------------------------------------

2013-02-13T10:36:43Z glouppe

The use case is when you want to combine several estimators together. It is natural for average-like ensembles, but makes no sense in boosted ensembles. In that perspective, I see "extending an estimator" as "combining" it with more base estimators.

-------------------------------------------------------------------------

2013-02-13T10:42:06Z amueller

So the setting is that you have trained some bagging estimators and want to combine them together, right?

In which setting do you want to do that except for parallelization? It is not so clear to me but maybe I'm overlooking something obvious.

-------------------------------------------------------------------------

2013-02-13T10:49:55Z glouppe

In case of Stacking the estimators might be completely different (say to you want to merge forests with svms).

(Indirectly, this could also be used to implement subsampling strategies or for monitoring the fitting process.)

-------------------------------------------------------------------------

2013-02-13T10:59:06Z amueller

I'm not sure I get the stacking example. I would have imagined that if we had a stacking interface, you could specify one estimator as the base estimator and another as the one on top.

-------------------------------------------------------------------------

2013-02-13T11:39:29Z glouppe

As I see it, the point of stacking is to combine the predictions of estimators of different nature. The more diverse they are, often the better.

-------------------------------------------------------------------------

2013-02-13T12:17:07Z amueller

Ok, so the base estimators would be different. But then we could also build this into the interface for stacking, right?

-------------------------------------------------------------------------

2014-01-09T07:45:51Z jwkvam

Resolved with #2570

-------------------------------------------------------------------------

2014-01-09T07:46:58Z glouppe

@jwkvam We recently agreed in #2570 to implement this feature using the `warm\_start` parameter. It is now implemented in GBRT. I'll try to update the forests with the same mechanism before the release.

-------------------------------------------------------------------------

2014-01-09T07:58:36Z jwkvam

@glouppe You're right, I forgot I had written this for any ensemble. But really I just wanted it for GBRT :) so in my haste, I decided this issue was resolved. If you like you can reopen it and close it when you are done, it doesn't matter to me.

-------------------------------------------------------------------------