Project: scikit-learn

2018-01-23T09:14:09Z jorisvandenbossche 63comments

Rethinking the CategoricalEncoder API ?

Web url: https://github.com/scikit-learn/scikit-learn/issues/10521

API url: https://api.github.com/repos/scikit-learn/scikit-learn/issues/10521

Based on some discussions we are having here and issues that are opened, we are having some doubts that `CategoricalEncoder` (https://github.com/scikit-learn/scikit-learn/pull/9151) was the good choice of name (and since it is not released yet we have some room for change).

So summary of how it is now:

- The class name `CategoricalEncoder` says what type of data it accepts (categorical data)

- The keyword argument `encoding` specifies \*how\* to encode those data

Currently we already have `encoding='onehot'|'onehot-dense'|'ordinal'`.

But what to do in the following cases:

- We want to add more encoding options (eg binary encoding, mean target encoding, unary encoding, ...). Do we keep adding those as new values for the `encoding` kwarg in the one big `CategoricalEncoder` class?

- We want to add an option specific to one of the encodings (eg for 'onehot' encoding to drop the first (redundant) column, or for 'ordinal' encoding base the order of the categories on the frequency, ...). The problem here is that we then need to add additional keyword arguments to `CategoricalEncoder` that are or are not active depending on what you passed for `encoding` kwarg, which is not the nicest API design.

For that last problem, we already had this with the `sparse=True/False` option, which was only relevant for 'onehot' and not for 'ordinal', and which we solved with having both 'onehot' and 'onehot-dense' encoding options and not a `sparse` keyword. But such an approach also does not scale.

Related to this, there is a PR to add a `UnaryEncoder` (https://github.com/scikit-learn/scikit-learn/pull/8652). There was a related discussion on the naming in that PR, as currently the name says \*how\* it encodes, not what type of data it gets (in the current design, it accepts already encoded integers, not actual categorical data. In that regard, to be consistent with CategoricalEncoder, it might better be named OrdinalEncoder because it needs ordinal data as input).

---

What are the options forward:

1) Keep things as we have it now in master, and be be OK with adding some new options to the single class (an important question which is hard to answer now, is how much new features we will want to add in the future).

2) Switch the naming scheme and have a bunch of 'categorical encoders' where the name says how it encodes (OnehotEncoder, OrdinalEncoder, and later maybe BinaryEncoder, UnaryEncoder, ...)

So it is a bit a trade-off of potential build up of number of classes vs number of keyword arguments in a single class.

---

One problem with the second approach (and one of the reasons we went with `CategoricalEncoder` in the first place, even before we added the multiple encoding options), is that there is already a `OnehotEncoder`, which has a different API than the `CategoricalEncoder`. And, there is not really a good other name we could use for the encoder that does one-hot encoding.

However, I think that, with some temporary ugly hacks, we could reuse the name, if we are OK with deprecating the current attributes (and I think we agree it are not the most useful attributes). The idea would be that if you fit the class with string data, you get the new behaviour, and if you fit the class with integer data, you get a deprecation warning indicating the default behaviour will change (and indicating which keyword to specify to get rid of the warning).

cc @jnothman @amueller @GaelVaroquaux @rth

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

2018-01-23T09:46:00Z ogrisel

Thanks for the summary @jorisvandenbossche. I think I am in favor of option 2: reuse `OneHotEncoder` class, deprecate the weird attributes and add a constructor parameter to select the behavior with a future warning that says that the default behavior will change but makes it easy to silence that warning just by passing a value for that option.

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

2018-01-23T11:27:45Z jnothman

The idea of reverting CategoricalEncoder makes me quite sad, but I think

you're right that future users would be less mystified by option 2. My main

concern is that we have tried implementing this as a change to OHE for a

long time and it never flew. Perhaps it would be good to attempt the

modifications to the OneHotEncoder docstring according to that proposed

change, so we can see if it looks sane.

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

2018-01-23T11:32:43Z GaelVaroquaux

+1 to what Joel said

‚Å£Sent from my phone. Please forgive typos and briefness.‚Äã

On Jan 23, 2018, 12:28, at 12:28, Joel Nothman <notifications@github.com> wrote:

>The idea of reverting CategoricalEncoder makes me quite sad, but I

>think

>you're right that future users would be less mystified by option 2. My

>main

>concern is that we have tried implementing this as a change to OHE for

>a

>long time and it never flew. Perhaps it would be good to attempt the

>modifications to the OneHotEncoder docstring according to that proposed

>change, so we can see if it looks sane.

>

>

>--

>You are receiving this because you were mentioned.

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

>https://github.com/scikit-learn/scikit-learn/issues/10521#issuecomment-359761818

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

2018-01-23T12:34:02Z jorisvandenbossche

> The idea of reverting CategoricalEncoder makes me quite sad

To be clear, it would not be a revert, it would be a refactor / rename that keeps all functionality!

But I also like the "CategoricalEncoder" name, that would indeed be sad.

That said, I will quickly try to do the changes to have an idea how possible it is to integrate this in OnehotEncoder.

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

2018-01-23T15:17:32Z jorisvandenbossche

OK, I opened a PR with a proof of concept: https://github.com/scikit-learn/scikit-learn/pull/10523.

It's not yet complete (no deprecation warnings and new attributes are not yet calculated in old behaviour).

The main API question is about the format of the input data.

So as a recap, there are \*\*two different ways we currently process the categorical data\*\*:

1) As actual, not yet encoded (integer or string), categorical data (how it is done in `CategoricalEncoder`) -> infer categories from the unique values in the training data

2) As integer, already encoded data (how it is done in the current `OneHotEncoder`) -> infer categories from the maximum value in the training data

The question is: \*\*do we find both cases worth supporting?\*\* Thus, in the potentially merged OneHotEncoder, do we keep the ability to do both, or do we fully deprecate and then remove the ability to process ordinal input?

If want the ability to process both, we can add a boolean keyword to specify the input data type (for now I use `encoded\_input=False/True`, but other ideas are `ordinal\_input`, ...)

For the deprecation period, we have to support both anyway, and also have to introduce a keyword to choose the behaviour (to be able to silence the warning and choose the new behaviour).

So in principle we could just keep the keyword afterwards.

Given that we want to handle both, an overview of how OneHotEncoder would work:

- for now `encoded\_input=None`, and we infer the default based on the data

- if int-like data (handled before by OneHotEncoder) `encoded\_input` is internally set to True and a deprecation warning is raised. If the user wants to keep the current behaviour, it can manually specify it as `OneHotEncoder(encoded\_input=True)` to silence the warning.

- if the input is not int-like, we set `encoded\_input` internally to False and without warning use the new behaviour (= the current CategoricalEncoder behaviour)

- in the future we change the default of `encoded\_input` from None to False (by default the new behaviour, also for int-like data)

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

2018-01-24T01:56:30Z jnothman

I'm still not sure what you're suggesting is the practical difference due to inferring categories from the max value.

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

2018-01-24T09:40:21Z jorisvandenbossche

@jnothman I suppose you acknowledge there \*can\* be a difference in practice? (the output you get depending on the data you have)

But whether this difference is important in practice, I don't know. That's where I would like to see feedback. Whether anybody actually \*wants\* this "max value"-based method, or whether we are fine with (in the future, after deprecation) only having the "unique values"-based method.

I think I personally would never need this max-value based method, but the OneHotEncoder has been like that for many years (for good reason or not?).

Actually deprecating the max-value based categorization would certainly make the implementation (after deprecation) simpler.

And if we choose for that route, I agree the option should rather be `legacy\_mode=True/False` rather than `encoded\_input`/`ordinal\_input`

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

2018-01-24T10:10:32Z jnothman

Remind me what the actual difference in output is, when n\_values='auto',

please? I had thought the active\_features\_ thing made them basically

identical, but I'm probably forgetting something.

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

2018-01-24T10:34:42Z jorisvandenbossche

Aha, that clarifies our misunderstanding :-)

I misunderstood how the current OneHotEncoder is actually working. Suppose you have one feature with values [2, 3, 5, 2]. I thought the current OneHotEncoder would have categories [0, 1, 2, 3, 4, 5] (while current CategoricalEncoder would have categories [2, 3, 5]). But you are right that the `active\_features\_` is also only [2, 3, 5], essentially making them the same with the default value of `n\_values='auto'`.

So it is only the case where you pass an integer to `n\_values` (like `n\_values=6` for categories=[0, 1, 2, 3, 4, 5] in the above case) to specify the number of categories that will actually be an API change (deprecated / removed).

And that will be easily be replacable by the user with `categories=range(6)`

Sorry for the confusion.

In that light, I think we even don't need the `legacy\_mode` option. We can just translate `n\_values=6` to `categories=range(6)` internally and raise a warning for that (but need to check this with the actual tests).

The other difference is the handling of unseen categories. With the current behaviour of the OneHotEncoder, if the unseen values are within the range(0, max), it will not raise an error even if `handle\_unknow='error'` (the default). But also that can be solved separately by in such a case raising a warning that the user should set `handle\_unknown='ignore'` manually to keep the existing behaviour.

The only feature we would loose is the distinction between unknown categories that are within the range(0, max) (by the current OneHotEncoder not regarded as 'unknown') and those that are bigger than that (> max, those are currently already regarded as unknown by the OneHotEncoder).

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

2018-01-24T11:22:25Z jnothman

no, that is the sort of thing we have tried before and it's just too

finicky. unless there is good reason to maintain current behaviour, we

should just have a legacy\_mode to slowly bring us to the future.

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

2018-01-24T11:24:52Z jorisvandenbossche

> no, that is the sort of thing we have tried before and it's just too finicky.

Can you clarify to which aspect this "no" refers?

To the fact that I think a `legacy\_mode` is not needed?

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

2018-01-24T20:06:46Z jnothman

yes, to the idea that you can just make something that is both backwards

compatible and what we want going forward

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

2018-01-24T22:32:37Z jorisvandenbossche

> yes, to the idea that you can just make something that is both backwards compatible and what we want going forward

That was not what I tried to suggest. I wanted to make clear that think it is possible to not have a `legacy\_mode` keyword, not by having it magically both backwards compat and what we want in the future, but by deprecating the behaviour of the existing keywords.

So to be concrete: a non-default value of `n\_values` can be deprecated and has to be replaced by `categories` specification. `handle\_unknow` in case of integer data should be set explicitly by the user to choose either full ignoring or full erroring instead of current mix (and otherwise deprecation warning is raised).

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

2018-01-25T00:42:09Z jnothman

so if I do .fit([[5]]).transform([[4]]), for which values of n\_values,

categories and handle\_umknown will that raise an error?

On 25 Jan 2018 9:32 am, "Joris Van den Bossche" <notifications@github.com>

wrote:

> yes, to the idea that you can just make something that is both backwards

> compatible and what we want going forward

>

> That was not what I tried to suggest. I wanted to make clear that think it

> is possible to not have a legacy\_mode keyword, not by having it magically

> both backwards compat and what we want in the future, but by deprecating

> the behaviour of the existing keywords.

>

> So to be concrete: a non-default value of n\_values can be deprecated and

> has to be replaced by categories specification. handle\_unknow in case of

> integer data should be set explicitly by the user to choose either full

> ignoring or full erroring instead of current mix (and otherwise deprecation

> warning is raised).

>

> ‚Äî

> You are receiving this because you were mentioned.

> Reply to this email directly, view it on GitHub

> <https://github.com/scikit-learn/scikit-learn/issues/10521#issuecomment-360296569>,

> or mute the thread

> <https://github.com/notifications/unsubscribe-auth/AAEz6-DrQWep22\_gs-hg9cC0u19B1\_PSks5tN6-HgaJpZM4RpUE8>

> .

>

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

2018-01-25T00:53:45Z jnothman

can we just make it so that during deprecation, categories must be set

explicitly, and legacy mode with warnings is otherwise in effect? Is that

what you are suggesting?

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

2018-01-28T23:00:37Z jorisvandenbossche

> can we just make it so that during deprecation, categories must be set explicitly, and legacy mode with warnings is otherwise in effect? Is that what you are suggesting?

Yes, it might be still missing case, but I \*think\* this is possible (will check by actual coding it next week).

The different 'legacy' cases:

- n\_values='auto' (the default)

- handle\_unknown='ignore' -> fine, no change in behaviour

- handle\_unknown='error' -> Problem, values in range are still ignored, values above range error

- Possible solution:

- in fit, if the range is consecutive => fine, no change in behaviour (for all people that now combined LabelEncoder with it, which is a typical use case I think)

- if this is not the case: raise deprecation warning that they have to set categories explicitly to keep this behaviour (and internally use legacy mode)

- n\_values=value

- this can be translated to categories=[range(value)] internally, and raise deprecation warning that user should do that themselves in the future

- in this case `handle\_unknown='error' / 'ignore'` work as expected

The deprecation warning in case of `n\_values='auto'` will only be raise in `fit` and not upon construction (which is not really ideal), but it is only in fit that we know that the user is passing it numeric data and not string data.

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

2018-01-28T23:30:40Z jnothman

we don't usually raise warnings until fit in any case so don't worry about

that.

that strategy sounds mostly good.

I'm not actually sure if we should be sniffing for strings in the data,

though. You basically want it to be: legacy mode is active if categories is

not set \*and\* if the data is all integers?

One question: if categories and n\_values parameters are their default, do

we publish categories\_? If n\_values is set explicitly, do we publish

categories\_?

On 29 Jan 2018 10:00 am, "Joris Van den Bossche" <notifications@github.com>

wrote:

> can we just make it so that during deprecation, categories must be set

> explicitly, and legacy mode with warnings is otherwise in effect? Is that

> what you are suggesting?

>

> Yes, it might be still missing case, but I \*think\* this is possible (will

> check by actual coding it next week).

>

> The different 'legacy' cases:

>

> - n\_values='auto' (the default)

> - handle\_unknown='ignore' -> fine, no change in behaviour

> - handle\_unknown='error' -> Problem, values in range are still

> ignored, values above range error

> - Possible solution:

> - in fit, if the range is consecutive => fine, no change in

> behaviour (for all people that now combined LabelEncoder with it, which is

> a typical use case I think)

> - if this is not the case: raise deprecation warning that

> they have to set categories explicitly to keep this behaviour (and

> internally use legacy mode)

> - n\_values=value

> - this can be translated to categories=[range(value)] internally,

> and raise deprecation warning that user should do that themselves in the

> future

> - in this case handle\_unknown='error' / 'ignore' work as expected

>

> The deprecation warning in case of n\_values='auto' will only be raise in

> fit and not upon construction (which is not really ideal), but it is only

> in fit that we know that the user is passing it numeric data and not string

> data.

>

> ‚Äî

> You are receiving this because you were mentioned.

> Reply to this email directly, view it on GitHub

> <https://github.com/scikit-learn/scikit-learn/issues/10521#issuecomment-361104495>,

> or mute the thread

> <https://github.com/notifications/unsubscribe-auth/AAEz6x8xnyZXBLij-DCC45JyYNf8pA5kks5tPPwXgaJpZM4RpUE8>

> .

>

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

2018-02-01T16:26:39Z jorisvandenbossche

> You basically want it to be: legacy mode is active if categories is not set \*and\* if the data is all integers?

Yes indeed (in practice it will more or less be the same)

> One question: if categories and n\_values parameters are their default, do we publish categories\_? If n\_values is set explicitly, do we publish categories\_?

I personally would already as much as possible provide the attributes of the new interface, even in legacy mode. So in both case I would calculate `categories\_` (even if it would be a bit more work)

---

So I tried to put the above logic in code (will push some updates to the PR), and I have one more question for the case of integer data when `n\_values` or `categories` is not set (typical case for 'legacy\_mode'). The problem lies in the fact that if the inferred categories are simply a consecutive range (0, 1, 2, 3, ... max), there is no difference between the new and old (legacy) behaviour, and we don't necessarily need to raise a deprecation warning.

Some possibilities to do in this specific case:

1) Detect this case (that the inferred categories are a consecutive range), and in that case don't raise a warning.

- This is possible to detect (with a little bit of extra code complexity) as we are already in fit anyhow

- I \*think\* this will be a common case when using OneHotEncoder with integer data, and a case where the user actually does not need to worry about our refactoring, so it would be nice to not bother him/her with a warning

2) Always raise a warning, and indicate in the warning message what to do if you are in such a case (in addition to an explanation what to do if you don't have a consecutive range):

- If they know they have only consecutive ranges as categories, they want to ignore the warning, so we can add to the warning message an explanation how to do this (add a code sample with filterwarnings they can copy paste)

- A potential advantage of this is that we can also add to the warning message that if they used the LabelEncoder to create the integers, they can now directly use OneHotEncoder (I think this currently is a typical usage pattern). That way, the warning will also go away

3) Always raise a warning but provide a keyword to silence it (eg `legacy\_mode=False`)

- If we find the advice to use a `filterwarnings` statement (see point 2 above) too cumbersome, we could also add a keyword to obtain the same result

- Disadvantage of this is introducing a keyword that will not be needed anymore in a few releases when the deprecations are cleaned up.

I am personally in favor of option 1 or 2. Using the LabelEncoder before OneHotEncoder seems to be a typical pattern (from a quick github search), and in those case you \*always\* have consecutive ranges, and there will never be a change in behaviour with the new implementation, so we shouldn't warn for it. On the other hand, if we warn we can point them to the fact that \*if\* they used LabelEncoder, they no longer need to do it. Which would be nice to actually give this advice explicitly.

The question is how frequently users have such consecutive integers as categories without having used LabelEncoder as the previous step ..

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

2018-02-01T16:39:45Z jorisvandenbossche

Hmm, one case I forgot is when you have integer inferred categories that are \*not\* consecutive (let's say [1,3,5]), but you want the new behaviour and not legacy behaviour (so in that case you cannot just ignore the warning, as that would handle unseen values differently in the transform step, i.e. values in between the range (eg 2) will not raise an error).

In case we don't provide the `legacy\_mode=False` keyword, the only way to obtain the new behaviour is by manually passing `categories=[1,3,5]`, which can be a slight inconvenience. That might be a reason to favor option 3 and give up my objection on introducing a temporary keyword `legacy\_mode=False` (but also not fully sure it is worth it, as this would be the only case\\* where such a keyword is actually needed)

\\* this only case = integer data with inferred categories that are not consecutive range, and where you cannot / don't want to set the categories manually or set handle\_unknown to ignore.

Sorry for all the long text, but it's quite complex :)

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

2018-02-01T21:01:53Z jnothman

We're only talking about the case where n\_values is unset, right?

I'm fine with 1., and it would not be any more expensive, since auto

already needs to examine the set of labels. I could also accept, for

simplicity, a variant of 3. that was just "OneHotEncoder running in legacy

mode. Set categories='auto' for slightly different behaviour without a

warning."

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

2018-02-01T21:08:46Z jorisvandenbossche

> We're only talking about the case where n\_values is unset, right?

Yes (the other case easily be translated in its equivalent `categories` value, with a nice deprecation warning, and without different in new and legacy behaviour)

> a variant of 3. that was just "OneHotEncoder running in legacy mode. Set categories='auto' for slightly different behaviour without a warning."

Ah, that sounds like a good idea! (irregardless of whether detecting the consecutive categories case or not). So we set in the code the default of `categories` to None (without changing the semantics of its default), so we know if the user set it explicitly, and in that way it is a nice way to indicate `legacy\_mode=False` without needing that extra keyword.

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

2018-02-01T21:38:33Z jnothman

Yes, but only if we want to warn every time someone uses it without passing

categories. It's the cheap implementation approach, but it might be

unnecessarily verbose for the users, which is why I would prefer 1 if it

can be done simply.

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

2018-02-07T16:58:11Z amueller

What fresh hell is this :-/

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

2018-02-07T17:01:18Z amueller

OR we could name the new one ``DummyEncoder`` ;) (though that is a bit conflicting with the DummyClassifier)

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

2018-02-07T17:01:37Z jorisvandenbossche

@amueller Don't read all of the above!

I was just planning to make a nice summary for new readers of the issue. The above discussion is overly complicated (also because I was still not fully understanding the current complex behaviour of OneHotEncoder ... :-))

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

2018-02-07T17:03:08Z jorisvandenbossche

> OR we could name the new one DummyEncoder ;)

I think @GaelVaroquaux was against that because "one-hot" is known to be this in more fields (and we already use 'Dummy' for other things in scikit-learn ...)

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

2018-02-07T17:03:28Z amueller

Redoing this for consistency in naming is not worth it imho. We are not consistent in naming anywhere. Could you summarize the discussions that lead to this?

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

2018-02-07T17:04:04Z amueller

I think "dummy" is what statisticians use and it's what pandas uses.

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

2018-02-07T17:05:31Z jorisvandenbossche

The top post is still accurate and worth a read, and it summarizes the reasoning for not keeping CategoricalEncoder (which does not mean that we need to use OneHotEncoder instead of eg DummyEncoder, that's a separate question)

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

2018-02-07T17:07:25Z amueller

I read the top post. That's what I referred to when I said "redoing this for consistency is not worth it".

> issues that are opened

Can you explain that?

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

2018-02-07T17:16:18Z jorisvandenbossche

> "redoing this for consistency is not worth it"

With consistency, are you pointing to the naming scheme of "what it accepts" vs "what it does" ? If so, that was only a minor reason. For me it is mainly a question of scalability in adding more features to a single class.

> issues that are opened

We had the issue about how to handle missing values (https://github.com/scikit-learn/scikit-learn/issues/10465), and for this you could want different behaviour for ordinal and one-hot encoding (or not all options are valid for both, ..). We also already have the existing `handle\_unknown` which is only relevant for one-hot encoding and not for ordinal. And there was https://github.com/scikit-learn/scikit-learn/issues/10518 about feature weighting for onehot encoding, but also not relevant for ordinal (this issue in the end was not a problem, as you can do the weighting with the ColumnTransformer transformer\_weights argument). And we also have the feature request to add something like `drop\_first` for one-hot, which is again not relevant for ordinal encoding.

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

2018-02-07T17:19:21Z amueller

I don't see how the proposed change would help with the missing values that much. And having incompatible options is something that happens often in scikit-learn. Not ideal, but also not a big deal imho.

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

2018-02-07T17:26:27Z jorisvandenbossche

> I don't see how the proposed change would help with the missing values that much.

It does not help \*as such\*, but it makes it less complex to have specific options specifically tailored to the different encoding types.

> And having incompatible options is something that happens often in scikit-learn. Not ideal, but also not a big deal imho.

Currently it is certainly still OK, there are not too many incompatible options (but also partly because I moved `sparse=True/False` into the `encoding` option). But the question is to what extent we want to expand the encoding functionality in scikit-learn in the future. Which is of course a difficult question to answer \*now\*.

We already have a PR for 'unary encoding'. Shouldn't this be rather added to CategoricalEncoder instead of adding a new class UnaryEncoder? And what if someone wants to add a 'binary encoding'? Or a '(mean) target encoder'?

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

2018-02-07T17:29:38Z amueller

The "mean target encoder" is ``CountTransformer``, there's a PR for that ;)

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

2018-02-07T17:31:18Z jorisvandenbossche

Do you have a link for that? Searching for "CountTransformer" does not give any results

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

2018-02-07T17:32:24Z amueller

Sorry, CountFeaturizer #9614

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

2018-02-07T17:42:07Z jorisvandenbossche

It's certainly related, but not exactly a mean target encoding. Also, it adds columns, not replaces, so will not yet work out of the box for string categorical data (but that is more feedback on that PR, not to discuss here).

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

2018-02-07T17:43:18Z amueller

Why is it not mean target encoding? But yeah let's not divert too much here ;)

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

2018-02-07T17:46:56Z jorisvandenbossche

So as a summary of the actual questions we need to answer (in this order!):

1. Do we keep the current `CategoricalEncoder`? If not, the idea is to split it in different classes, one class for each type of encoding (currently 'onehot' and 'ordinal' encoding).

2. If we split in multiple classes, we could (ideally?) use OneHotEncoder for the 'onehot' encoding, but this class already exists. So, do we integrate the new 'onehot' encoding (which supports strings and has different parameters) in the existing OneHotEncoder class? Or do we choose another name? (eg DummyEncoder)

3. If we choose to integrate into the existing OneHotEncoder, are we OK with the following consequences: we deprecate a bunch of the keywords/attributes of OneHotEncoder, and a specific usecase (automatically ignoring unseen values within the \*range\* of seen values) will not be possible anymore after deprecation period.

Most of the discussion above was about question 3 (the complex details of how to integrate CategoricalEncoder(encoding='onehot') into OneHotEncoder). But let's first agree on a decision for the first 2 questions.

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

2018-02-07T20:57:25Z jnothman

the other factor for me is that everyone thinks the current auto mode in

OneHotEncoder is weird. its implementation converting coo to csr is also

weird. it deserves a redesign. and telling people "if you want to one hot

encode strings, go to CategoricalEncoder instead" is awkward, because OHE

is already intended for categoricals...

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

2018-02-07T22:51:49Z amueller

hrm. I guess we kept OneHotEncoder because it's more efficient when it can be used.... Ideally we would get rid of all the weird behaviors. I kinda had wanted to deprecate it but then we didn't...

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

2018-02-07T22:55:32Z jorisvandenbossche

> I kinda had wanted to deprecate it but then we didn't...

In my POC PR (https://github.com/scikit-learn/scikit-learn/pull/10523), I deprecated almost everything of OneHotEncoder, except its name ...

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

2018-02-07T23:39:43Z jnothman

It's not much more efficient. And if LabelEncoder had fast paths for ints

in range [0, n\_values-1], if justified, that would be good enough.

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

2018-02-11T23:55:36Z jnothman

@amueller, are you persuaded by the issue that we ultimately want different additional parameters (e.g. drop\_first, nan handling) depending on the encoding, and that justifies having a different discrete encoder for each encoding format?

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

2018-02-23T20:38:23Z amueller

I'll try to look at this in the spring break in two weeks, ok? not sure if I'll have time before that :-/

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

2018-02-24T12:22:43Z lesshaste

I hope this isn't the wrong place to ask but what does the current implementation do with tables that are mixed categorical and non-categorical within one column? Taking the example from https://github.com/pandas-dev/pandas/issues/17418

Consider the dataframe `df = pd.DataFrame([{'apple': 1, 'pear':'a', 'carrot': 1}, {'apple':'a', 'pear':2, 'carrot':3}, {'apple': 2, 'pear':3, 'carrot':1}, {'apple': 3, 'pear':'b', 'carrot': 1}, {'apple': 4, 'pear':4, 'carrot': 1}])` which equals:

```

apple carrot pear

0 1 1 a

1 a 3 2

2 2 1 3

3 3 1 b

4 4 1 4

```

DictVectorizer gives exactly what I need in this case.

```

from sklearn.feature\_extraction import DictVectorizer

enc = DictVectorizer(sparse = False)

enc.fit\_transform(df.to\_dict(orient='r'))

```

This gives:

```

array([[ 1., 0., 1., 0., 1., 0.],

[ 0., 1., 3., 2., 0., 0.],

[ 2., 0., 1., 3., 0., 0.],

[ 3., 0., 1., 0., 0., 1.],

[ 4., 0., 1., 4., 0., 0.]])

```

We can see the features names of the columns with:

```

enc.feature\_names\_

['apple', 'apple=a', 'carrot', 'pear', 'pear=a', 'pear=b']

```

It would be great if the new CategoricalEncoder had an option to do the same.

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

2018-02-24T14:38:15Z jnothman

I don't think we intend to handle that kind of mixed case

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

2018-02-24T21:14:16Z lesshaste

That‚Äôs a shame. One simple sub case is where a column is numerical but has some missing values. A simple solution is to convert the NaNs into empty strings and then use DictVectorizer as in my example above. This effectively creates a new feature for when the value is missing but leaves the numerical values unchanged otherwise. I have found this a very useful technique.

Will the new CategoricalEncoder be able to do something similar?

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

2018-02-25T05:25:46Z jnothman

we've considered allowing users to have NaN treated as a separate category

or similar. but that's not the same as handling arbitrary numeric values as

different from strings.

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

2018-02-25T07:10:36Z lesshaste

That sounds good.

You are right there are two use cases. Let me explain a particular example of where treating numeric values as different from strings has been useful for me. It may be that there is a better solution.

Say you have an integer numeric feature which takes a large range of values. However you suspect that for some small values, the precise value is significant. For larger values you suspect this isn‚Äôt the case. A simple thing to do is to convert all small values to strings, run DictVectorizer as above and then perform feature selection or just use your favorite classifier directly.

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

2018-02-25T09:34:39Z jnothman

So you're using it for a non-linear discretisation? The next release is

likely to include a fixed-width discretizer, but following on from a log

transform or a quantile transform it should act quite similar to what you

want... But the log transform might alone be sufficient in your setting.

On 25 February 2018 at 18:10, lesshaste <notifications@github.com> wrote:

> That sounds good.

>

> You are right there are two use cases. Let me explain a particular example

> of where treating numeric values as different from strings has been useful

> for me. It may be that there is a better solution.

>

> Say you have an integer numeric feature which takes a large range of

> values. However you suspect that for some small values, the precise value

> is significant. For larger values you suspect this isn‚Äôt the case. A simple

> thing to do is to convert all small values to strings, run DictVectorizer

> as above and then perform feature selection or just use your favorite

> classifier directly.

>

> ‚Äî

> You are receiving this because you were mentioned.

> Reply to this email directly, view it on GitHub

> <https://github.com/scikit-learn/scikit-learn/issues/10521#issuecomment-368288727>,

> or mute the thread

> <https://github.com/notifications/unsubscribe-auth/AAEz60cmjwlDVKGyXc6oPyIC9oLbptSgks5tYQdvgaJpZM4RpUE8>

> .

>

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

2018-02-25T09:37:34Z lesshaste

@jnothman Yes in a sense except with a twist. Say I suspect that some of the values from 1...1024 are meaningful. That is 22 indicates something specific which is quite different from 21 or 23. Taking logs won't help here. But I want to leave all the values over 1024 as numerical as I don't think those specific values mean much.

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

2018-02-25T09:42:19Z jnothman

It sounds like you know too much about your variable for a generic

transform to be the sort of thing you need.

On 25 February 2018 at 20:37, lesshaste <notifications@github.com> wrote:

> @jnothman <https://github.com/jnothman> Yes in a sense except with a

> twist. Say I suspect that some of the values from 1...1024 are meaningful.

> That is 22 indicates something specific which is quite different from 21 or

> 23. Taking logs won't help here. But I want to leave all the values over

> 1024 as numerical as I don't think those specific values mean much.

>

> ‚Äî

> You are receiving this because you were mentioned.

> Reply to this email directly, view it on GitHub

> <https://github.com/scikit-learn/scikit-learn/issues/10521#issuecomment-368295895>,

> or mute the thread

> <https://github.com/notifications/unsubscribe-auth/AAEz65bOdVB6k7rCAcgLBYz\_NslxXWV0ks5tYSnggaJpZM4RpUE8>

> .

>

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

2018-02-25T10:00:35Z lesshaste

@jnothman To be a little clearer, I don't know that 22 is significant. I just suspect that \*some\* values are but I don't know which ones or how many there are. I have found the "convert to a string" and then DictVectorizer method to be very useful for discovering which these are.

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

2018-02-26T09:19:56Z jorisvandenbossche

@lesshaste For the issue about NaNs as separate category, see https://github.com/scikit-learn/scikit-learn/issues/10465

If you want to further discuss the specific non-linear discretization or mixed numeric/string encoding, feel free to open a new issue. But would like to keep this one focused on the original issue, i.e. the \*naming\* and organisation in different classes of the CategoricalEncoder/OneHotEncoder.

> I'll try to look at this in the spring break in two weeks, ok? not sure if I'll have time before that :-/

@amueller that's fine. I won't have time the coming two weeks to work on the PR that is blocked by this anyway. After that I should also have time again to work on it.

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

2018-03-20T10:29:42Z jorisvandenbossche

@amueller did you have time to give this a look?

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

2018-03-29T10:03:03Z jorisvandenbossche

@amueller are you ok with that I go ahead with working on the PR to split CategoricalEncoder in OrdinalEncoder and OneHotEncoder (and with deprecating current arguments of OneHotEncoder) ?

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

2018-04-26T20:52:17Z amueller

Sorry for being absent. Seems ok, but can you maybe give me two weeks so I can actually review? Thanks!

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

2018-05-14T09:07:27Z jorisvandenbossche

@amueller no problem, for me the same :-)

But, I am now planning to look at this again. So if you could give this a look that would be welcome. I have some work to do on the PR (https://github.com/scikit-learn/scikit-learn/pull/10523), so don't review that yet in detail (you can look at it to have an idea of what we propose however).

I think the main question I want to see answered before I put a lot of time in it, is if you are OK with splitting up CategoricalEncoder into multiple classes, and in that case, if you are OK with re-using OneHotEncoder (which means deprecating some of its current (strange) features). Those questions are summarized in https://github.com/scikit-learn/scikit-learn/issues/10521#issuecomment-363851328 and https://github.com/scikit-learn/scikit-learn/issues/10521#issuecomment-364802471.

(and once we agree on that part, there is still a lot to discuss about the actual implementation in the PR :))

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

2018-05-17T18:41:17Z jorisvandenbossche

I updated the PR https://github.com/scikit-learn/scikit-learn/pull/10523, ready for review

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

2018-05-19T18:49:15Z amueller

I'll cautiously say I'm back ;)

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

2018-05-28T12:37:09Z kuraga

IMHO the most important thing is a universal API (i.e. parameters and bbehavior patterns) for all of encoders we discuss

P.S. https://github.com/scikit-learn-contrib/categorical-encoding ?

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

2018-05-28T13:15:02Z jorisvandenbossche

In the `category\_encoders` package, all encoders have a `cols` argument, similar to the `categorical\_features` in the old OneHotEncoder (although it does not accept exactly the same kind of values). See eg http://contrib.scikit-learn.org/categorical-encoding/onehot.html

So that is related to the current discussion we are having in https://github.com/scikit-learn/scikit-learn/pull/10523 about deprecating `categorical\_features` or not.

For the rest I think there are not really conflicting keywords (they have some others specific to dataframes which we won't add to sklearn at this point). The naming for OneHotEncoder and OrdinalEncoder at least is consistent with the `category\_encoders` package.

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

d OrdinalEncoder at least is consistent with the `category\_encoders` package.

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