slundberg / **shap**

New issue

Output value in binary classification task is outside [0, 1] range #29

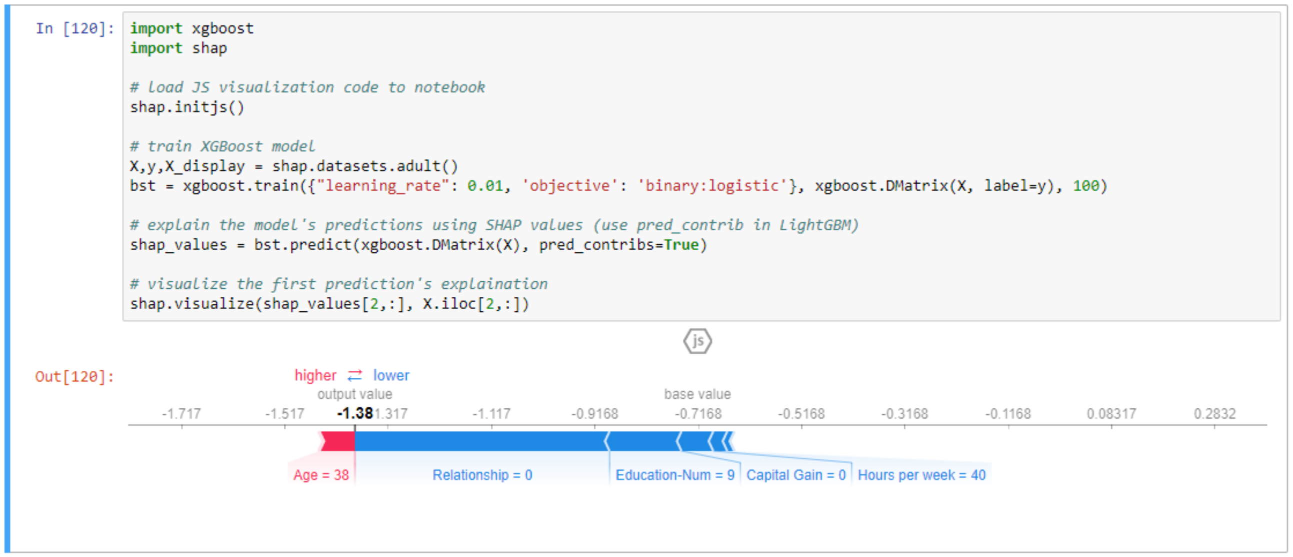
Closed **asstergi** opened this issue on Feb 1, 2018 · 30 comments

asstergi commented on Feb 1, 2018

Hi **@slundberg**,

I've been playing with a binary classification task using XGBoost and I noticed an unexpected (for me at least) behaviour. I replicated it using the `adult` dataset you're providing.

So, after training a binary classfication XGBoost model and plotting the SHAP values for a case, I'm getting the following:



```
In [120]: import xgboost
          import shap

          # load JS visualization code to notebook
          shap.initjs()

          # train XGBoost model
          X,y,X_display = shap.datasets.adult()
          bst = xgboost.train({"learning_rate": 0.01, 'objective': 'binary:logistic'}, xgboost.DMatrix(X, label=y), 100)

          # explain the model's predictions using SHAP values (use pred_contrib in LightGBM)
          shap_values = bst.predict(xgboost.DMatrix(X), pred_contribs=True)

          # visualize the first prediction's explaination
          shap.visualize(shap_values[2,:], X.iloc[2,:])

Out[120]:
```

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

9 participants

Both the base value and the output value are outside the [0, 1] range. Is this the expected bahavior? If so, how can someone interpret this?

slundberg commented on Feb 1, 2018 Owner

This is because the XGBoost Tree SHAP algorithm computes the SHAP values with respect to the margin not the transformed probability. So the values you are seeing are log odds values (what XGBoost would output if `pred_margin=True` were set).

The visualize function accepts a link function to transform the x-axis from log odds to probabilities:
`shap.visualize(shap_values[2,:], X.iloc[2,:], link=shap.LogitLink())`

Note that this option was always there for the visualize function that took an explainer object (from the model agnostic examples), but I just pushed an update to expose it for the matrix interface you are using.

slundberg closed this on Feb 1, 2018

asstergi commented on Feb 5, 2018 Author

@slundberg
You could possibly apply the same change to the `visualize()` function here as well:
https://github.com/slundberg/shap/blob/master/shap/plots.py#L511

slundberg commented on Feb 5, 2018 Owner

Ha! Good point, done.

MHonegger commented on Mar 9, 2018

Hi **@slundberg**,

first of all, I would like to thank you for developing such a great tool! I am using it in my master thesis, to explain the outcomes in a predictive maintenance use case.

I have ran into some difficulties however, related to the question above.
So what I would like to do is to extract the transformed shapley values, that are used in force_plot, when we use link= 'logit'.

I am not sure why I am experiencing such difficulties, as I know we could hypothetically just apply the expit() function to the shapley values and base value, to backtransform them.
When I apply this function, the base value for the predicted class is correctly retransformed, as you can see in the screenshot it should be 0.08566 (and then see the output of the second screenshot, last value).
However, all shapley values are way to large... So I basically do not understand why when I take the sum of the shapley values plus the base value... it does not add up to the output value (which should be around 0.98).

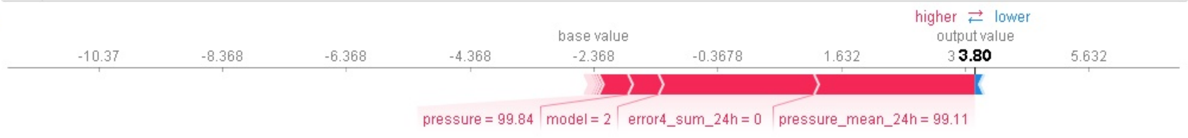
Note that I have a multiclass classification problem with label classes 0 (about 98% of all objects) to 5 (so label 1, 2, 3 and 4 constitute machine failures, while 0 means no failure).

I would be extremly grateful for some help!
Best,
Milo

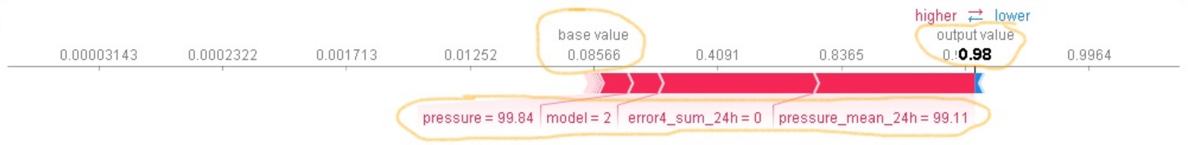
```
In [107]: np.set_printoptions(suppress=True) #Surpress scientific Notation
          xgb_logit_shap_values[360,60:80]

Out[107]: array([ 0.01162159,  0.00899264,  0.47205073, -0.00027627,  0.01060358,
                  0.04407672,  2.6003191 ,  0.0397529 , -0.00506782,  0.01393221,
                 -0.1431232 ,  0.02023939,  0.05023585,  0.008871  ,  0.00180488,
                  2.5115986 ,  0.01494313,  0.49469295,  0.01129662, -2.367785  ],
                dtype=float32)
```

```
In [110]: from scipy.special import expit
          expit(xgb_logit_shap_values[360,60:80])

Out[110]: array([0.50290537, 0.50224817, 0.61586905, 0.49993092, 0.5026509 ,
                 0.5110174 , 0.93088216, 0.5099369 , 0.49873304, 0.503483  ,
                 0.46428013, 0.50505966, 0.5125563 , 0.5022177 , 0.5004512 ,
                 0.92495096, 0.5037357 , 0.62121135, 0.5028241 , 0.08566247],
                dtype=float32)
```

```
In [78]: #Plot
         shap.force_plot(xgb_logit_shap_values[360,60:80], xgb_X_test, link = 'identity') #shap.LogitLink()

Out[78]:
```



```
In [77]: #Plot
         shap.force_plot(xgb_logit_shap_values[360,60:80], xgb_X_test, link = 'logit') #shap.LogitLink()

Out[77]:
```



asstergi commented on Mar 9, 2018 Author

@MHonegger The values below the plot refer to the raw feature values, not to the SHAP values.

MHonegger commented on Mar 9, 2018

@asstergi yes that's clear, but that's not what I need, what I need is the numbers that e.g. pressure_mean_24h contributes about lets say 0,40 and error4_sum_24h contributes about 0,38 or something like that... In the scale of the output value!

slundberg commented on Mar 9, 2018 • edited

Owner

@MHonegger Good question. As you discovered, you can't just transform the values individually. Putting a non-linear function on the output of another function transforms the SHAP values of the function in very subtle ways. If you could just transform them like you were trying, then I could write a high speed exact version for deep learning models. But you can't, so instead we approximate them by a linearization in the Deep SHAP approach mentioned in the paper.

I would suggest you use the same approximation here, it works by transforming the base value as you did, then scaling the other SHAP values so they sum to the transformed model output (when added to the base value). This corresponds to the exact SHAP values for a first order taylor approximation of the logit in the transformed function.

It might even be worth making some handy utility function for anyone else look to do the same thing.

slundberg commented on Mar 9, 2018

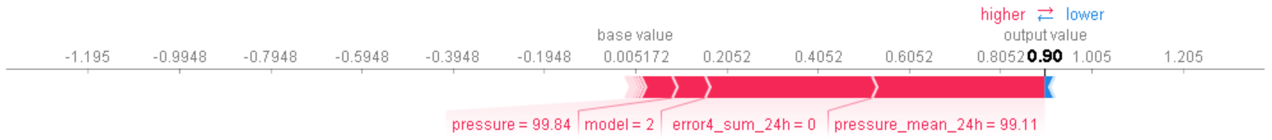
Owner

I should also point out that adding and subtracting the margin of a logistic function corresponds to adding and subtracting bits of information to the log odds, while adding and subtracting probabilities is not quite as mathematically natural a space to work in (though we have a better intuition of what a 10% change means than a 3 bit change).

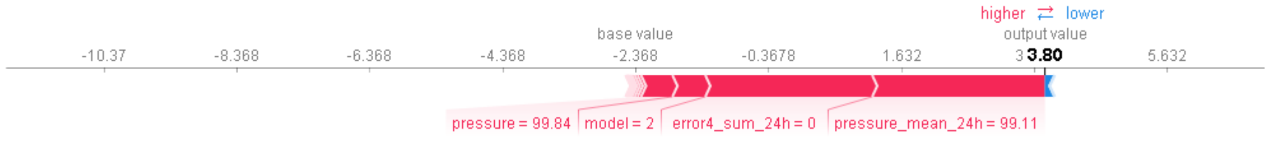
MHonegger commented on Mar 12, 2018

@slundberg Thank you very much for the detailed and helpful answer!!!
I have been testing around and I have done what you suggested, however, with a small change, though I am not sure if it is correct to do what I did?
Basically as you said, I took the model prediction probability as the output value, however, as a base value I chose to take the relative occurence of the label in the test dataset (which is consistent with what the implementation for the Kernel Shap does?). So if e.g. one label has a relative occurence of 97%, then that would be the base value... If the prediction probability for that object was 99,99%, then that would be my output value. The distance between the basis and output is scaled such that the proportions remain the same as in the non-scaled "native" output of the shap algorithm. I've attached one exemplary screenshot with the three variants (using Logit link, Identity and the transformed one, which I implemented).
It would be amazing if you could quickly comment if/why using the transformed base value instead of the relative occurence of that label class is the correct way to go?
Thank you very much, I highly appreciate it!

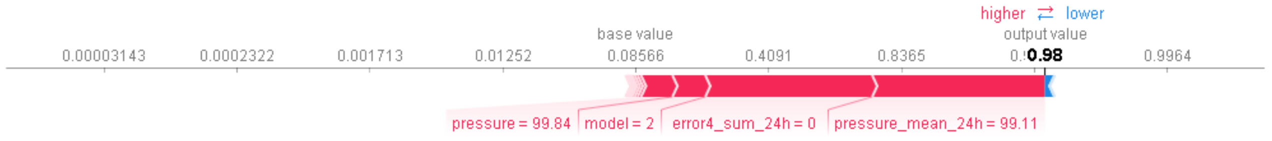
```
#Plot
shap.force_plot(xgb_shap_transform_scale(360), xgb_X_test) #shap.LogitLink()
```



```
#Plot
shap.force_plot(xgb_shap_values[360,60:80], xgb_X_test, link = 'identity') #shap.LogitLink()
```



```
#Plot
shap.force_plot(xgb_shap_values[360,60:80], xgb_X_test, link = 'logit') #shap.LogitLink()
```



slundberg commented on Mar 12, 2018 Owner

The difference between using the label frequency in the test set as the base value and using the transformed base value is two fold:

1. You are getting the expected value of the label in the test set instead of the expected value of the model output in the training set (SHAP values are defined with respect to the expected value of the model output).

2. The base value is the expected value of the margin of the model output, if you instead average in the probability space (instead of the margin space) you get a different answer.

Ultimately if you want SHAP values with respect to probabilities instead of bits, you have to make some kind of approximation since the logit makes it more complicated.

If it were me, I would find the expected value of the model's output on the training data and use that as the base probability value. Then proportionally scale the SHAP values to stretch between that base value and the current model output. This approach is a linear approximation of the logit that is adjusted to match the true base rate of the model in the probability space.

MHonegger commented on Mar 21, 2018

@slundberg Sorry for getting back late and thank you so much for your answer!

It is of course not a good idea to use the label frequency of the test set... If anything, then the frequency of the label classes in the training set (to also prevent leaking and to be consistent with the definition w.r.t. the model output as you mentioned).

I will try to build the function analogously to the approach mentioned in the Deep SHAP paper, i.e. using the transformed base value with the logit function and then stretching the shap values between that and the model output, as you mentioned before. If I am not able to do it, I will use the label frequency of the training set as base values (as you suggested).

In any way, I will also post the function here, so that other people who run into the same or similar problem can use it (I am not experienced enough to pushing something directly onto your repository if that's even possible -.-').

Best,
Milo

MHonegger commented on Mar 21, 2018

Here is the function I successfully used for the above-described task.

Notes:

- The function assumes that you only pass it an array of the shapley values of the class you wish to explain (so if you e.g. have a multiclass problem with 5 classes, and the object you wish to explain belongs to class 3, then only pass the array of shapley values and base value of class 3)
- The model_prediction variable is the actual prediction probability for that particular object that you got as a model output (e.g. your XGB model is 99,96% certain that the above object actually belongs to class 3, then that number will be your model_prediction)

```
def xgb_shap_transform_scale(shap_values, model_prediction):

    #Compute the transformed base value, which consists in applying the logit function to the
base value
    from scipy.special import expit #Importing the logit function for the base value
transformation
    untransformed_base_value = shap_values[-1]
    base_value = expit(untransformed_base_value )

    #Computing the original_explanation_distance to construct the distance_coefficient later on
    original_explanation_distance = sum(shap_values[0, -1])

    #Computing the distance between the model_prediction and the transformed base_value
    distance_to_explain = abs(model_prediction - base_value)

    #The distance_coefficient is the ratio between both distances which will be used later on
    distance_coefficient = original_explanation_distance / distance_to_explain
```

```
        #Transforming the original shapley values to the new scale
        shap_values_transformed = shap_values / distance_coefficient

        #Finally resetting the base_value as it does not need to be transformed
        shap_values_transformed [-1] = base_value

        #Now returning the transformed array
        return shap_values_transformed
```

Once you have transformed your shapley values with the above function, you can call the plot function with your shap_values_transformed as follows:

```
shap.force_plot(shap_values_transformed, test_set)
```

Hope it helps anyone with the same task. If you have any questions let me know!
Best,
Milo

👍 2

slundberg commented on Mar 21, 2018 Owner

Thanks for sharing! If I get a chance I'll merge a version of that in at
some point
…

👍 1

germayneng referenced this issue on May 7, 2018

returns probability instead of log odds #75 Closed

Toekan commented on May 9, 2018

> This is because the XGBoost Tree SHAP algorithm computes the SHAP values with respect to the margin not the transformed probability. So the values you are seeing are log odds values (what XGBoost would output if pred_margin=True were set).

Thanks for all the explanations in this thread, helped a lot!

But I'm still stuck with this answer, is there a reason why shap can't calculate SHAP values for the output probabilities (or change in probability compared to the base expected probability), rather than log values? (Theoretically, efficiency, etc ...) I don't see an immediate reason in the Tree SHAP paper (but then again, the algorithm takes some time to understand. :)) and your notebook Understanding Tree SHAP for Simple Models also seems to directly create SHAP values for output probabilities.

Toekan referenced this issue on May 9, 2018

Shap values are log-odds when using xgboost with objective='binary:logistic' Closed
#79

slundberg commented on May 9, 2018 Owner

Tree SHAP works by computing the SHAP values for trees. In the case of XGBoost, the output of the trees are log-odds that are then summed over all the trees and then sent through a logistic function to get a probability. In the Understanding Tree SHAP for Simple Models examples the sklearn trees directly output probabilities. So it all depends on what the model you are using outputs from the trees.

Rescaling the log-odds to create a probability is a good approximation, but exactly computing the values after a logistic transform gets messy and I don't know how to do it efficiently (not for a lack of thinking about it). Note that for visualization you can also just change the axis using the `link` option for the `force_plot`. Changing the axis really makes the most sense in my opinion since adding and subtracting probabilities is typically not a great idea (this is why logistic regression exists).

👍 2

Toekan commented on May 10, 2018

Thanks for your quick answer! So the reason I was missing is the fact that Tree SHAP calculates the shap values per tree to then adds them. Then of course you have to work in an additive space like log-odds. Thanks for the explanation! This means that indeed I will rescale the log-odds instead.

I was eager to have SHAP values related to probabilities because I want to calculate metrics out of it, e.g. the mean absolute value of the SHAP value per feature to create an overall feature importance, or means of feature interactions to rank interactions over the sample population. Because the problems I work with are typically very imbalanced and we care mainly about the samples that are given a high probability, assigning an importance related to probability rather than log-odds is probably seems
more useful. (e.g. we care more about a sample that goes from 1% change to 10% than one that drops to 0.1%). On top of that, many people I have to talk to don't like logs too much. :)

👍 1

Toekan commented on May 10, 2018 • edited

While trying to transform from log scale to probabilities, I realised I don't fully understand what the base value stands for. My assumption was that this represented \phi_{0}, so the model output when everything is set missing. Considering Tree SHAP takes both branches of a split when splitting on a missing value, this means \phi_{0} should represent the average value (of the raw log-odds output) over all possible splits and thus over all samples? Just like, I think, you mean here:

> The base value is the expected value of the margin of the model output

Unfortunately when I take the average of the raw output of XGboost over all the training samples used, I get a different value than what the last column of the shapley values gives me.

EDIT: forgot to mention, the base_score input parameter of XGboost is 0.5, so the initial expected value is 0.

What's wrong with my reasoning?

In general thanks a lot for trying to make your model accessible, this and other issues and the notebooks have been incredible helpful in trying to understand how to use SHAP!

slundberg commented on May 10, 2018

Owner

@Toekan Glad that makes sense!

...as for the base value mismatch I think that gets at a detail of how XGBoost weighs its samples. The proportion of samples that went down each branch in XGBoost is recorded by summing up the hessians of the samples. So in other words, XGBoost implicitly weighs samples differently, depending on the hessian of the loss at that sample (high hessian -> high weight). The current implementation of SHAP uses these hessian sums (since the simple counts are not saved) to measure the proportion of samples that went down each branch (as does every other metric in XGBoost). This means just taking the average margin output on the training dataset will not match the base value from SHAP unless you have a simple squared loss (which has a constant hessian). This is something specific to XGBoost's way of measuring sample weights, and could be worth changing in XGBoost sometime. LightGBM uses regular sample counts, and so do sklearn models.

Does that help?

👍 1

Toekan commented on May 14, 2018

That definitely helps, thanks for the clear explanation!

slundberg referenced this issue on May 17, 2018

How is the "BaseValue" for TreeShap computed? #90 Closed

skamkar referenced this issue on Jun 22, 2018

shap values scale differ for model type #128 Closed

trungnt37 commented on Jul 9, 2018

@slundberg
Hi Slundberg, how to caculate shap_values with model with ntree_limit = model.best_ntree_limit?
shap.force_plot(shap_values[0,:], X.iloc[0, :], link=shap.LogitLink()) do not return probability of best model?
Does that help? Thanks you

slundberg commented on Jul 9, 2018 Owner

@trungnt37 Just use the `tree_limit` parameter, for example: `explainer.shap_values(X, tree_limit=500)`

👍 1

trungnt37 commented on Jul 9, 2018

Great, thank you very much for helpful answer!

jmmonteiro commented on Jul 27, 2018

Hi @slundberg,

First of all, thank you for your work on SHAP, it an amazingly useful tool!

I use it to compute the shapley values, and then plot them using my own matplotlib scripts (unfortunately I can't use the default plotting functions). I've been trying to get the transformed values, and not the log odds values, is there any utility/function which will allow me to do it?

I tried the script posted by here, but I made a few changes to make it compatible with the new version of SHAP.

```
from scipy.special import expit #Importing the logit function for the base value transformation
def shap_transform_scale(shap_values, expected_value, model_prediction):

    #Compute the transformed base value, which consists in applying the logit function to the base v
    expected_value_transformed = expit(expected_value)
    #Computing the original_explanation_distance to construct the distance_coefficient later on
    original_explanation_distance = sum(shap_values)

    #Computing the distance between the model_prediction and the transformed base_value
#     distance_to_explain = abs(model_prediction - expected_value_transformed)
    distance_to_explain = model_prediction - expected_value_transformed

    #The distance_coefficient is the ratio between both distances which will be used later on
    distance_coefficient = original_explanation_distance / distance_to_explain

    #Transforming the original shapley values to the new scale
    shap_values_transformed = shap_values / distance_coefficient

    return shap_values_transformed, expected_value_transformed
```

Where `expected_value` comes from `shap.TreeExplainer(model).expected_value`.

It's a bit of a hack, so I'm not sure if it is correct. Also, I've noticed that the original line `distance_to_explain = abs(model_prediction - expected_value_transformed)` would cause the signs of some shapely values to flip. I assume this is not the intended behavior, so I removed the `abs()`.

Best,
Joao

slundberg commented on Jul 27, 2018 • edited

Owner

@jmmonteiro Thanks for sharing an update to this approach. You are right, it is a bit of a hack, and it can fail in cases where the `distance_coefficient` is zero.

This problem is the same issue faced in DeepExplainer, where we have SHAP values for a component that we then want to send through a non-linear transformation. It turns out that in order to avoid the `distance_coefficient = 0` problem, you have to send the samples through one at a time (rather than take the SHAP values for an expectation over the whole dataset). This is not possible with the current version of Tree SHAP, but we are thinking of how to combine some ideas from Deep SHAP and Tree SHAP so we can actually support the kind of transformation you are talking about. It would still be an approximation, but at least a well-motivated one that doesn't have divide by zero issues.

P.S. What you have posted seems like the best stop gap for now from just reading the code.

👍 2

ReinforcedMan commented on Sep 12, 2018

Hi all,

I take this opportunity to relaunch this discussion, as this log-odds space is a bit of a pain for me.

Kernel SHAP is quite slow and imprecise (with the two approximations, assuming feature independance and only computing expectations on a small background dataset).

Tree SHAP returns values in the log-odds space, which is in my experience (and as said above) impossible to reliably linearly scale back when the prediction on the sample is close to the average prediction.

Is there any fast and exact solution that returns SHAP values in the probability space ? Even if it's model specific. I didn't find the time yet to try DeepSHAP, maybe I should look there ?

Ps: Congrats on the work done so far, really appreciate SHAP

slundberg commented on Sep 12, 2018 Owner

@MPeter-29 I understand why having the value in probability space would be helpful, but I should note that I think it makes more sense to consider feature effects "adding" together in log-odds space. I personally prefer the approach taken by `force_plot` when using a logit link function where we just transform the x-axis labels and not the actual values.

That being said, we have been working on how to take some ideas from the Deep SHAP approximation and apply them to trees. This will allow us to transform the SHAP values into probability space without the instability problems discussed above. We have a python proof of concept but it will take a bit more work to iron out the details and write a C++ version. I am about to be on travel for the rest of the month so it will be a bit before that happens.

👍 5

alipeles commented on Nov 12, 2018 • edited

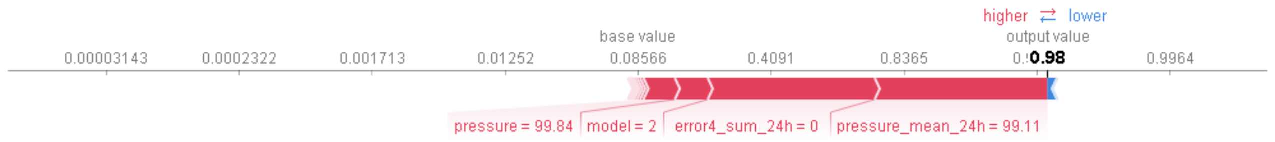
Apologies in advance if I’ve just misunderstood this, but a couple of things are puzzling me here.

First, per **@slundberg**'s post, force_plot draws each SHAP value the same size, regardless of whether link is ‘identity’ or ‘logit’ and, instead, changes the scale of the x-axis, which essentially becomes expit of the original scale, centered around expit(base_value). That’s a non-linear scaling of the SHAP values, while the solutions proposed by **@jmmonteiro** and **@MHonegger** are purely linear. I can’t tell from the discussion whether this is an intentional difference?

More puzzling for me, though, is that the scale on force_plot drawn with link=‘logit’ seems to change the relative contributions of the SHAP values.

Looking at the image below, originally posted by **@MHonegger** , the bar for error4_sum_24h is close in size to the bar for pressure_mean_24h (presumably a little bit smaller, since it’s on the left). But, the scale of the x-axis where error4_sum_24h lies is more than twice the scale where pressure_mean_24h lies. So, it looks like error4_sum_24h actually contributes much more than pressure_mean_24h.

```python
#Plot
shap.force_plot(xgb_shap_values[360,60:80], xgb_X_test, link = 'logit') #shap.LogitLink()
```



slundberg commented on Nov 18, 2018 • edited ▾ Owner

@alipeles A few quick thoughts:

One important point about just changing the axis is that it does not change the base value to be the mean of the probabilities, but instead the probability of the mean of the log-odds (as you say). Transforming into the probability space would require changing that base value. So it is not a proper conversion to the space.

You are right that equal length bars traverse different amount of probabilities, but they do traverse the same change in log odds space. Plotting probabilities on the x-axis is really just an easy way to do the log-odds to probability conversion. In a very real sense when a tree pushes the margin over by a fixed amount in the log odds space it can mean a different change in probability depending on what other evidence is in place for the current prediction (otherwise you go would out of the 0-1 range).

Finally I should note that master now has preliminary support for conversion to probability space if you look at the doc string in TreeExplainer. It is still getting cleaned up though so until we push it as a release use at your own risk :)

slundberg referenced this issue on Jan 19

Interpretation of Base Value and Predicted Value in SHAP Plots #352 Open

torgyn commented on Jan 21

> Finally I should note that master now has preliminary support for conversion to probability space if you look at the doc string in TreeExplainer. It is still getting cleaned up though so until we push it as a release use at you own risk :)

@slundberg I noticed that a parameter 'model_output' has been added to shap.TreeExplainer() in tree.py that allows a user to toggle between 'probability', 'margin' and 'log_odds' to be then passed to the output_transform_codes(). Unfortunately, the 'probability' option does not seem to be implemented or there is a bug in _cext.dense_tree_shap()? I am using shap v0.27 and aiming to obtain shap values in probability space for xgboost.sklearn.XGBClassifier trained with objective='binary:logistic'. When I set model_output= 'probability' I get identical result to when the model_output= 'margin'.

slundberg commented on Jan 21 Owner

@torgyn could you try again on master? It is working for me (and a bug with the expected_value calculation in that case is fixed on master but not v0.28)

slundberg added a commit that referenced this issue on Jan 21

`Allow XGBoost node weights to be overridden` … ✗ `a3de87a`