In this project my aim is to predict the tags of a select corpus of posts from the *Hinduism Stack Exchange*. Where the user tags have been coalesced into topics relating to primary Hindu scripture, including: *The Mahabharata*, *Ramayana*, *Vedanta*, *Vedas*, and *Puranas*. Where the Hinduism Stack Exchange describes itself as [“a question and answer site for followers of the Hindu religion and those interested in learning more about Hinduism.”]( <https://hinduism.stackexchange.com/>) It functions in the same way as other StackExchange forums, with users creating Q&A posts under the supervision of a moderation team that also creates wikis and removes off-topic or illicit content. Stack Exchange users can create their own tags subject to the StackExchange [guidelines]( <https://meta.stackexchange.com/help/privileges/create-tags>) and any user can include up to 5 different tags in one post. The dataset that we will be working with constitutes about 1/3rd of the StackExchange’s Q&A posts, with 40,370 unique tags and with more than 85% of all posts having multiple tags.

My motivation here is to practice fitting a general linearized model (GLM) to text data using tf-idf and to learn a little about the Hindu canon. It also gives me an excuse to use SQL to query our dataset and to use the `tidymodels` package for R.

Anyone with a casual familiarity of Hindu texts would probably ask “*where is the Upanishads or the Bhagavid-gita- why isn’t the Mahabharata and Ramayana grouped under an equally broad category like history or Itihasa?”* This is due to the distribution of user tags in the *Hinduism Stack Exchange*. Where the *Ramayana* and *Mahabharata*, as well as the *Vedas*, are overrepresented compared to other topics.

[insert a table of tags, something with datatable maybe?]

In the interest of capturing as large of the dataset as possible, and to avoid having to bootstrap resample, the less prevalent tags were grouped together. This consists of the eighteen *mahapuranas*, and the *Vedas*, while the *Upanishads*, *Bhagavad-gita*, and *Brahma-sutras* are under the *Vedanta*.

### Preprocessing and Feature Engineering

### Finalized `workflow()` and metrics

Now that we have our optimal penalty selected let’s finalize our workflow and fit it to our testing data. We’ll then collect our performance metrics with the aptly named `collect\_metrics()` function.

From the `tibble` output above, model accuracy is 0.79 and our `roc\_auc`,

### ROC-AUC Plot

Let’s see how well our model was able to predict class tags by plotting a confusion matrix.

The diagonal line on the left side is well populated, this means that our model \_generally\_ predicted the correct tag for each class. While the off-diagonal numbers indicate where our model misclassified data. The right-hand plot omits the correct predictions, this is to better visualize where our model breaks down. As mentioned earlier, the nebulous distinction between the works making up the \_Vedanta\_ from the \_Vedas\_ led to a degree of confusion. Though this isn’t as prominent as the instances where the \_Purans\_ are misclassified.

Next we use the `hardhat` package to cast our data to a \_sparse matrix\_, this will improve the performance of Lasso regressions in `glmnet`. This function creates a blueprint for our workflow to cast our feature rich \_data.frame\_ into a sparse matrix where most elements are zero. After that we create our initial workflow and add our preprocessing recipe, multinomial regression, and our sparse matrix recipe blueprint.

### Sparse matrix

We now use the `hardhat` package to specify a `blueprint` to cast our data to a \_sparse matrix\_, in order to take advantage of the innate sparsity of our data. A sparse matrix is, as the name implies, a matrix where most elements are zero, as most documents (in this case Q&A posts) do not contain most words. This is intended to improve computation time for our model- as sparse matrices drop the zero values in our data. An added benefit of using `glmnet` for our model is its ability to handle sparse input-matrices and, as defined here, “[at its core] is a set of Fortran subroutines, which make for very fast execution.”

### Hindu\_wf

Here we specify a \_model workflow\_ that allows us to easily bundle our preprocessor and model specification together. This will come in handy later on when we go to pass our preprocessor to `tune\_grid()` to output our performance metrics (e.g. accuracy or specificity). Plus making easier to modify our workflow in the future and for fitting our testing data.

### Glmnet

The next step is to specify a multinomial regression model fit using the `glmnet` engine. We set the Lasso penalty parameter (`mixture = 1`) to specify a pure Lasso model.

In this step we specify a multinomial regression model using the `multinom\_reg()` function included in the `tidymodels` package. We set the computational engine to use `glmnet` and specify a pure Lasso model by setting `mixture = 1`. The lasso regularized model uses a regularization method that also performs variable selection. It does this by determining how much of a \_penalty\_, denoted by λ, to apply to our features (sometimes going all the way to zero). This means that our high-dimensional feature space can be condensed down to a select group of important variables.

### ROC\_AUC

The ROC curve plots the true positive rate against false positives, with an AUC closer to 1 indicating a well performing model and an AUC closer to 0.5 (represented by the dashed line) indicating the model does no better than guessing. In this case, we see that each class prediction did more or less the same, achieving an average ROC AUC of 0.95. It confirms what we learned from our confusion matrix earlier, that \_*Vendata*\_ is more susceptible to misprediction than the others. But, otherwise, the classes move together through the different thresholds.

### Puranas

References to the *shatpatha brahamana*,

### Conclusions

I suspect that, due to having `step\_downsample()` come after `step\_tf-idf()` in our preprocessing recipe, our model now heavily weighs terms from the \_*Mahabharata\_* above the others. As it is the most populous of our classifiers. Otherwise, any issues arising in our model is likely due to the way in which tags were coalesced into broader topics. There is considerable diversity in the topics touched on in these posts and this is reflected in their tags. In many instances, references to Hindu scripture is only very tangentially related to the topic at hand. These sorts of post can have multiple other tags with varying levels of specificity that, if included, would provide a much clearer picture. Though, this could \_*also\_* throw our predictive model off, as the inclusion of more classifiers generally [lowers model performance]( <https://smltar.com/mlclassification.html#mlmulticlass>).

Altogether, this was meant to be an exercise in predictive modelling using `glmnet` and *Natural Language Processing (NLP)*. As a result, I didn’t explore additive features, for instance I could’ve created dummy variables for different *days of the week* or normalized post *scores* using a `step\_YeoJohnson()` function. Nor did I include any other classification models (or even a null model) to test for baseline performance- which reminds me of a humorous [(though not entirely relevant)]( http://varianceexplained.org/images/sliced/iq\_meme.png) meme. There were also different NLP components I didn’t touch on, the most apparent to me being word stemming. I had tested stemming and found it to have no real impact. In the end, this was an exercise in working with an \_alien\_ dataset that required me to learn about Hindu literature as well as practice supervised machine learning with text data.