Juice Recommending System for Diseases

Mohammed Saifuddin
1640272 - 5CMS

Mohammed Sameeruddin
1640273 - 5CMS

Purusharth Saxena
1640207 - 5CMS

Introduction

This is a simple experimental project in machine learning, that implements the `TensorFlow API` to built a model for recommending juices to certain ubiquitous diseases like, cold, fever and sore-throat. Recommending system essentially uses a filtering algorithm that brings all closely related units together from a dataset. There are certain practices which are dependent on the form of data, for our dataset, we implemented a neural network that *classifies* diseases and then recommend the corresponding juice(s) for the cure of same.

Our dataset mainly comprises of 3 labels namely - `cold`, `fever`, `soarthroat`. Most of the juices do intersect at curing more than one disease, for example, Tomato juice cures both sore-throat and cold, but whereas Grapefruit juice just cures cold. Due to this intersection, we get 7 labels considering all the combinations. Below table shows labels of the dataset and respective numerical code.

Label	**Code**
`cold`	0
`cold-fever`	1
`cold-soarthroat`	2
`cold-soarthroat-fever`	3
`fever`	4
`soarthroat`	5
`soarthroat-fever`	6

We have a visualization of the `sizes` for each label corresponding to their label names. In the graph, $x-$axis shows label-values that were coded and $y-$axis shows the respective sizes of each different label. With this bar graph we can comprehend that, `cold` and `fever` comprise a greater concentration in our data. Labels like `cold-fever` and `cold-soarthroat-fever` have the least number compared from the rest. Every column of the dataset is of numeric form except the `juice-names`. A machine learning model works smoothly only if the data is comprised of numbers, since we don't have numbers, we have *integer-encode* that categorical columns. Integer encoding is a process of encoding text data into discrete numbers where in model does not face any impeding consequences during learning.

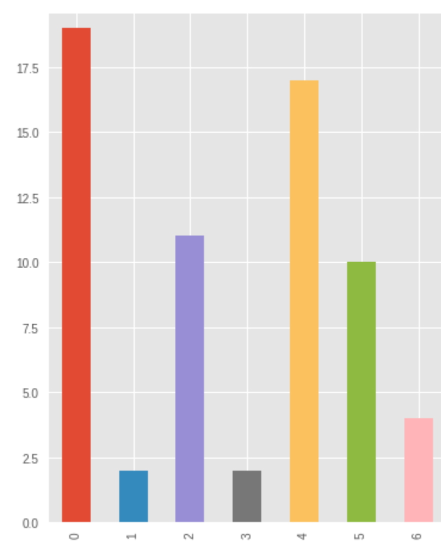


Figure 1: Sizes of each label

Building the Model

Before building a model, we have ensure that every *feature* in the data must be in *numeric-column*, this is because in real scenario not all datasets are numeric. In this dataset, `juice-names` columns are categorical, so we have to encode them into numbers.

Once after encoding we constructed *feature-columns* and an *input pipeline function* called `my-input-fn` to channelize the data to the model as per the convenience of `TensorFlow Estimator` class.

Construction of Feature Columns

Feature columns play a radical role while channelizing the data, to the model. They enable transformation of raw-data into formats that `Estimators` API can use. `Estimators` is an eminent class of `TensorFlow` where all *classification* and *regression* algorithms are precoded. Data can be of cleaved into two aspects, `numeric` and `non-numeric`. To create a feature-column, we invoke `tf.feature_column` module. We have two types of columns,

- *Dense* - This column comprises of `numeric_column`, `indicator_column`, `embedding_column`.
- *Categorical*: This has `categorical_column_with_identity`, `categorical_column_vocabulary_file`, `categorical_column_vocabulary_list`, `crossed_column`.

For our model, we had to only use `tf.feature_column.numeric_column(features_in_the_data)`.

Input Function Pipeline

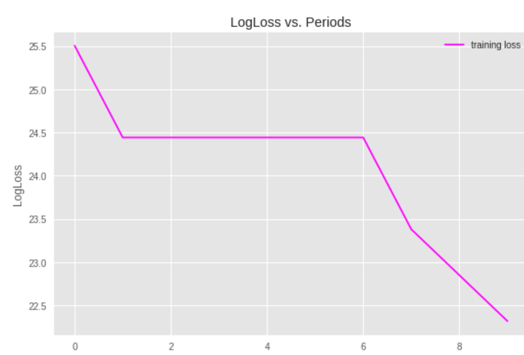
Input function is a pipeline function that takes examples from the data. Examples include both `features` and `labels`. With this function, we `slice` the data from tuple form, `(feature, labels)` and then do `batching` and `shuffling` of the data.

Training the Model

In this aspect, we instantiate an `object` of `tf.estimator.DNNClassifier()`. This is actually a deep neural network classifier to train the model with the data. This whole strand is so crucial. The object that is instantiated would be configured with an `optimizer` named `tf.train.GradientDescentOptimizer()`. This takes a parameter `learning_rate`, that can be any floating-point number. This parameter is referred as *tuning-parameter*, which is tuned most of the time by a practitioner to revamp the performance. The object that was created, takes a parameter named, `input_fn` which will be an implementation of pipeline function. Since `TensorFlow` is `Python` based, it is easy to create anonymous functions called `lambda`. We create a lambda function that invokes `my_input_fn` and pass it to the `input_fn` parameter.

With this we have a model, that has access to the data via, pipeline. Now we train it in a loop such that, the *weights* are chosen that minimizes the `loss` function $L(x)$. Loss is defined as the difference between `true` function $f(x)$ and `hypothesis` function $h(x)$. Hypothesis function generalizes the true function. Gradually, the model reduces `loss` and learns to generalize the data-points. On the left, we have table of all tuning-

Tuning-Parameters	Value
learning-rate	0.005
batch-size	3
steps	500
hidden-units	$[8, 7, 6, 4]$



parameters and their respective value. On the right we have `loss` gradually decreasing as the model started learning the data-points. Since we had only `65` units of data, the accuracy result that we obtained for this model was, `0.3538` or 35.3%.