**Data Transformation - Learning with Counts**

Learning with counts is an efficient way to create a compact set of dataset features that are based on counts of the values. You can use the modules in this category to build a set of counts and features. Later, you can update the counts and the features to take advantage of new data or merge two sets of count data.

**About count-based featurization**

The basic idea of count-based featurization is that by calculating counts, you can quickly and easily get a summary of what columns contain the most important information. The module counts the number of times a value appears and then provides that information as a feature for input to a model.

Imagine that you’re validating a credit card transaction. A crucial piece of information is where this transaction came from. One of the most common encodings of the transaction origin is the postal code. However, there might be as many as 40,000 postal codes, zip codes, and geographical codes to account for. Does your model have the capacity to learn 40,000 more parameters? If you give it that capacity, do you have enough training data to prevent it from overfitting?

If you have good data, with lots of samples, such fine-grained local granularity can be powerful. However, if you have only one sample of a fraudulent transaction, from a small locality, does it mean that all of the transactions from that place are bad, or that you don’t have enough data?

One solution is to learn with counts. Instead of introducing 40,000 more features, you can observe the counts and proportions of fraud for each postal code. By using these counts as features, you get information about the strength of the evidence for each value. Moreover, by encoding the relevant statistics of the counts, the learner can use the statistics to decide when to change their approach and instead use other features to get the information.

Count-based learning is attractive for many reasons. With count-based learning, you have fewer features, which requires fewer parameters. Fewer parameters make for faster learning, faster prediction, smaller predictors, and less potential to overfit.

**How count-based features are created**

A basic example can help demonstrate how count-based features are created and applied. Suppose you have the following table like this, with labels and inputs. Each case (or row or sample) has a set of values in columns. In this example, the values are A and B.

|  |  |
| --- | --- |
| Label column | Input Value |
| 0 | A |
| 0 | A |
| 1 | A |
| 0 | B |
| 1 | B |
| 1 | B |
| 1 | B |

These are the steps you take to create count-based features:

* For a specific set of values, find all the other cases in that dataset that have the same value. In this case, there are three instances of A and four instances of B.
* Count the class membership of each value as a feature in itself. In this case, you get a small matrix: there are two cases where A = 0; one case where A = 1; one case where B = 0; and three cases where B = 1.
* Based on this matrix, you get a variety of count-based features. These include a calculation of the log-odds ratio and the counts for each target class. The table in the next section displays the data.

**Build Counting Transform**

This describes how to use the Build Counting Transform module in Machine Learning Studio (classic), to analyze training data. From this data, the module builds a count table as well as a set of count-based features that can be used in a predictive model.

A count table contains the joint distribution of all feature columns, given a specified label column. Such statistics are useful in determining which columns have the most information value. Count-based featurization is useful because such features are more compact than the original training data, but capture all the most useful information. You can use the module parameters to customize how the counts are transformed into the new set of count-based features.

After generating counts and transforming them into features, you can save the process as a transformation for re-use on related data. You can also modify the set of features without having to generate a new set of counts, or merge the counts and features with another set of counts and features.

The ability to re-use and re-apply count-based features is useful in scenarios such as these:

* New data becomes available to improve the coverage or balance of your dataset.
* Your original counts and features were based on a very large dataset that you don’t want to re-process. By merging the counts you can update with new data.
* You want to ensure that the same set of count-based features is applied to all datasets that you are using in your experiment.

**How to configure Build Counting Transform**

You can create a count-based feature transformation directly from a dataset, and re-run it each time you run an experiment. Or, you can generate a set of counts, and then merge it with new data to create an updated count table.

* Create count-based features from a dataset

Start here if you have not created counts before. You use the Build Counting Transform module to create count tables and automatically generate a set of features.

This process creates a feature transformation that you can apply to a dataset, using the Apply Transformation module.

* Merge counts and features from multiple datasets

If you have already generated a count table from a previous dataset, generate counts on just the new data, or import an existing count table created in an earlier version of Machine Learning. Then, merge the two sets of count tables

This process creates a new feature transformation that you can apply to a dataset, using the Apply Transformation module.

**Create count-based features from a dataset**

* In Machine Learning Studio (classic), add the Build Counting Transform module to your experiment. You can find the module under Data Transformation, in the category Learning with Counts.
* Connect the dataset you want to use as the basis for our count-based features.
* Use the Number of classes option to specify the number of values in your label column.

1. For any binary classification problem, type 2.
2. For a classification problem with more than two possible outputs, you must specify in advance the exact number of classes to count. If you enter a number that is less than the actual number of classes, the module will return an error.
3. If your dataset contains multiple class values and the class label values are non-sequential, you must use Edit Metadata to specify that the column contains categorical values.

* For the option, The bits of hash function, indicate how many bits to use when hashing the values.

It is generally safe to accept the defaults, unless you know that there are many values to count and a higher bit count might be needed.

* In The seed of hash function, you can optionally specify a value to seed the hashing function. Setting a seed manually is typically done when you want to ensure that hashing results are deterministic across runs of the same experiment.
* Use the Module type option to indicate the type of data that you will be counting, based on the storage mode:

1. Dataset: Choose this option if you are counting data that is saved as a dataset in Machine Learning Studio (classic).
2. Blob: Choose this option if your source data used to build counts is stored as a block blob in Windows Azure storage.
3. MapReduce: Choose this option if you want to call Map/Reduce functions to process the data.

To use this option, the new data must be provided as a blob in Windows Azure storage, and you must have access to a deployed HDInsight cluster. When you run the experiment, a Map/Reduce job is launched in the cluster to perform the counting.

For very large datasets, we recommend that you use this option whenever possible. Although you might incur additional costs for using the HDInsight service, computation over large datasets might be faster in HDInsight.

* After specifying the data storage mode, provide any additional connection information for the data that is required:

1. If you are using data from Hadoop or blob storage, provide the cluster location and credentials.
2. If you previously used a Import Data module in the experiment to access data, you must re-enter the account name and your credentials. The Build Counting Transform module accesses the data storage separately in order to read the data and build the required tables.

* For Label column or index, select one column as the label column.

A label column is required. The column must already be marked as a label or an error is raised.

* Use the option, Select columns to count, and select the columns for which to generate counts.

In general, the best candidates are high-dimensional columns, together with any other columns that are correlated with those columns.

* Use the Count table type option to specify the format used for storing the count table.

1. Dictionary: Creates a dictionary count table. All column values in the selected columns are treated as strings, and are hashed using a bit array of up to 31 bits in size. Therefore, all column values are represented by a non-negative 32-bit integer.In general, you should use this option for smaller data sets (less than 1 GB), and use the CMSketch option for larger datasets. After selecting this option, configure the number of bits used by the hashing function, and set a seed for initializing the hash function.
2. CMSketch: Creates a count minimum sketch table. With this option, multiple independent hash functions with a smaller range are used to improve memory efficiency and reduce the chance of hash collisions. The parameters for hashing bit size and hashing seed have no effect on this option.

* Run the experiment.

The module creates a featurization transform that you can use as input to the Apply Transformation module. The output of the Apply Transformation module is a transformed dataset that can be used to train a model.Optionally, you can save the transform if you want to merge the set of count-based features with another set of count-based features. For more information, see Merge Count Transform.

**Export Count Table**

This describes how to use the Export Count Table module in Machine Learning Studio (classic). The Export Count Table module is provided for backward compatibility with experiments that use the deprecated Build Count Table and deprecated Count Featurizer modules.

When you use the new Build Counting Transform module to create count-based features, the module outputs both a featured dataset and a transform that creates features from counts. By using the Export Count Table module, you can separate the count-based features output by this newer module into count metadata and a count table. These output formats were used by earlier, now deprecated modules:

For general information about count tables and how they are used to create features, see Learning with Counts.

For all new experiments, we recommend that you use the following modules:

* Build Counting Transform
* Modify Count Table Parameters
* Merge Count Transform

**How to configure the Export Count Table**

In Machine Learning Studio (classic), open the experiment where you want to use the imported count table.

* Locate the saved count transformation, and add it to the experiment.
* Connect the output of the saved count transformation (labeled transformation) to Export Count Table.
* Add the Count Featurizer (deprecated) module to the experiment, and connect it to the two outputs of Export Count Table.
* The Count Featurizer (deprecated) module requires an additional input, for the dataset you want to featurize. Connect the dataset to apply the saved transformation to outputs.
* Set any necessary parameters for Count Featurizer (deprecated), including the label column, the count columns, the columns to featurize, and the features to output.

You must select a subset of the columns that were originally selected for the counting transformation. However, the Export Count Table module does not provide the list of these columns, so you should review the original experiment and make a note of which columns were used. If you select a column that was not used when creating the transformation, an error is raised.

**Import Count Table**

The purpose of the Import Count Table module is to allow customers who created a table of count-based statistics using an earlier version of Machine Learning to upgrade their experiment. This module merges the existing count tables with new data.

**How to configure the Import Count Table**

* In Machine Learning Studio (classic), open an experiment that contains a count table created using the deprecated Build Count Table module.
* Add the Import Count Table module to the experiment.
* Connect the two outputs of the Build Count Table (deprecated) module to the matching input ports of the Import Count Table.

If you have another dataset of counts that you want to merge with the imported count table, connect it to the rightmost input for the Import Count Table module.

* Use the Counting type option to specify where and how the count table is stored:

1. Dataset: The data used to build counts is saved as a dataset in Machine Learning Studio (classic).
2. Blob: The data used to build counts is stored as a block blob in Windows Azure storage.
3. MapReduce: The data used to build counts is stored as a blob in Windows Azure storage.

After specifying the data storage mode, you may need to provide additional connection information for the data, even if you previously used a Import Data module in the experiment to access data. That is because the Count Featurizer (deprecated) module accesses the data storage separately in order to read the data and build the required tables.

* Use the Count table type option to specify the format and storage mode of the table used to store counts.

1. Dictionary: Uses a dictionary count table.

All column values in the selected columns are treated as strings, and are hashed using a bit array of up to 31 bits in size. Therefore, all column values are represented by a non-negative 32-bit integer.

1. CMSketch: Uses a table saved in the count minimum sketch table.

With this format, multiple independent hash functions with a smaller range are used to improve memory efficiency and reduce the chance of hash collisions.

In general, you should use the Dictionary option for smaller data sets (<1GB), and use the CMSketch option for larger datasets.

* Run the experiment.
* When complete, right-click the output of the Import Count Table module, select Save as Transform, and type a name for the transformation. When you do this, the merged count tables and any featurization parameters you might have applied are saved in a format that can be applied to a new dataset.

**Merge Count Transform**

To combine two sets of count-based features. By merging two sets of related counts and features, you can potentially improve the coverage and distribution of the features.

Learning from counts is particularly useful in large data sets with high-cardinality features. The ability to combine multiple datasets into count-based feature sets without having to reprocess the datasets makes it easier to gather statistics on very large datasets and apply them to new datasets. For example, count tables can be used to collect information over terabytes of data. You can re-use those statistics to improve the accuracy of predictive models on small data sets.

To merge two sets of count-based features, the features must have been created using tables that have the same schema: that is, both sets must use the same columns, and have the same names and data types.

**How to configure Merge Count Transform**

* To use Merge Count Transform, you must have created at least one count-based transformation, and that transformation must be present in your workspace. If you saved a count-based transformation from a different experiment, look in the Transforms group. If you created the transformation in the current experiment, connect the outputs of the following modules:

1. Build Counting Transform. Creates a new count-based transformation from source data.
2. Modify Count Table Parameters. Takes an existing count transformation as an input and outputs an updated transformation.
3. Import Count Table. This module supports backward compatibility with older experiments that used count-based learning. If you used Import Count Table to analyze the distribution of values in a dataset, and then converted the values to features using the deprecated Count Featurizer module, use Import Count Table to convert the results to a transformation.

* Add the Merge Count Transform module to the experiment, and connect a transformation to each input.
* If you do not want the second dataset to be weighted equally with the first, specify a value for Decay factor. The value that you type indicates how the set of features from the second transformation should be weighted.

For example, the default value of 1 weights both sets of features equally. A value of .5 means that the features in the second set would have half the weight of those in the first set.

* Optionally, add an instance of the Apply Transformation module, and apply the transformation to a dataset.

**Modify Count Table Parameters**

To change the way that features are generated from a count table.In general, to create count-based features, you use Build Counting Transform to process a dataset and create a count table, and from that count table generate a new set of features.

However, if you have already created a count table, you can use the Modify Count Table Parameters module to edit the definition of how the count data is processed. This lets you create a different set of count-based statistics based on the existing data, without having to re-analyze the dataset.

**How to configure Modify Count Parameters**

* Locate the transformation you want to modify, in the Transforms group, and add it to your experiment.

You should have previously run an experiment that created a count transformation.

1. To modify a saved transform: Locate the transformation, in the Transforms group, and add it to your experiment.
2. To modify a count transformation created within the same experiment: If the transformation has not been saved, but is available as an output in the current experiment (for example, check the output of the Build Counting Transform module), you can use it directly by connecting the modules.

* Add the Modify Count Table Parameters module and connect the transformation as an input.
* In the Properties pane of the Modify Count Table Parameters module, type a value to use as theGarbage bin threshold.

This value specifies the minimum number of occurrences that must be found for each feature value, in order for counts to be used. If the frequency of the value is less than the garbage bin threshold, the value-label pair is not counted as a discrete item; instead, all items with counts lower than the threshold value are placed in a single "garbage bin".

If you are using a small dataset and you are counting and training on the same data, a good starting value is 1.

* For Additional prior pseudo examples, type a number that indicates the number of additional pseudo examples to include. You do not need to provide these examples; the pseudo examples are generated based on the prior distribution.
* For Laplacian noise scale, type a positive floating-point value that represents the scale used for introducing noise sampled from a Laplacian distribution. When you set a scale value, some acceptable level of noise is incorporated into the model, so the model is less likely to be affected by unseen values in data.
* In Output features include, choose the method to use when creating count-based features for inclusion in the transformation.

1. CountsOnly: Create features using counts.
2. LogOddsOnly: Create features using the log of the odds ratio.
3. BothCountsAndLogOdds: Create features using both counts and log odds.

* Select the Ignore back off column option if you want to override the IsBackOff flag in the output when creating features. When you select this option, count-based features are created even if the column doesn’t have significant count values.
* Run the experiment. You can then save the output of Modify Count Table Parameters as a new transformation, if desired.