tf.contrib.factorization.KMeansClustering

Class **`KMeansClustering`**

Inherits From: `Estimator`

Defined in `tensorflow/contrib/factorization/python/ops/kmeans.py`.

An Estimator for K-Means clustering.

Properties

`config`

`model_dir`

`model_fn`

Returns the model_fn which is bound to self.params.

Returns:

The model_fn with following signature: `def model_fn(features, labels, mode, config)`

`params`

Methods

`__init__`

```
__init__(
    num_clusters,
    model_dir=None,
    initial_clusters=RANDOM_INIT,
    distance_metric=SQUARED_EUCLIDEAN_DISTANCE,
    random_seed=0,
    use_mini_batch=True,
    mini_batch_steps_per_iteration=1,
    kmeans_plus_plus_num_retries=2,
    relative_tolerance=None,
    config=None
)
```

Creates an Estimator for running KMeans training and inference.

This Estimator implements the following variants of the K-means algorithm:

If `use_mini_batch` is False, it runs standard full batch K-means. Each training step runs a single iteration of K-Means and must process the full input at once. To run in this mode, the `input_fn` passed to `train` must return the entire input dataset.

If `use_mini_batch` is True, it runs a generalization of the mini-batch K-means algorithm. It runs multiple iterations, where each iteration is composed of `mini_batch_steps_per_iteration` steps. Each training step accumulates the contribution from one mini-batch into temporary storage. Every `mini_batch_steps_per_iteration` steps, the cluster centers are updated and the temporary storage cleared for the next iteration. Note that: * If `mini_batch_steps_per_iteration=1`, the algorithm reduces to the standard K-means mini-batch algorithm. * If `mini_batch_steps_per_iteration = num_inputs / batch_size`, the algorithm becomes an asynchronous version of the full-batch algorithm. However, there is no guarantee by this implementation that each input is seen exactly once per iteration. Also, different updates are applied asynchronously without locking. So this asynchronous version may not behave exactly like a full-batch version.

Args:

- `num_clusters`: An integer tensor specifying the number of clusters. This argument is ignored if `initial_clusters` is a tensor or numpy array.
- `model_dir`: The directory to save the model results and log files.
- `initial_clusters`: Specifies how the initial cluster centers are chosen. One of the following:
    - a tensor or numpy array with the initial cluster centers.
    - a callable `f(inputs, k)` that selects and returns up to `k` centers from an input batch. `f` is free to return any number of centers from `0` to `k`. It will be invoked on successive input batches as necessary until all `num_clusters` centers are chosen.
    - `KMeansClustering.RANDOM_INIT`: Choose centers randomly from an input batch. If the batch size is less than `num_clusters` then the entire batch is chosen to be initial cluster centers and the remaining centers are chosen from successive input batches.
    - `KMeansClustering.KMEANS_PLUS_PLUS_INIT`: Use kmeans++ to choose centers from the first input batch. If the batch size is less than `num_clusters`, a TensorFlow runtime error occurs.
- `distance_metric`: The distance metric used for clustering. One of:
    - `KMeansClustering.SQUARED_EUCLIDEAN_DISTANCE`: Euclidean distance between vectors `u` and `v` is defined as `||u - v||_2` which is the square root of the sum of the absolute squares of the elements' difference.
    - `KMeansClustering.COSINE_DISTANCE`: Cosine distance between vectors `u` and `v` is defined as `1 - (u . v) / (||u||_2 ||v||_2)`.
- `random_seed`: Python integer. Seed for PRNG used to initialize centers.
- `use_mini_batch`: A boolean specifying whether to use the mini-batch k-means algorithm. See explanation above.

- `mini_batch_steps_per_iteration`: The number of steps after which the updated cluster centers are synced back to a master copy. Used only if `use_mini_batch=True`. See explanation above.
- `kmeans_plus_plus_num_retries`: For each point that is sampled during kmeans++ initialization, this parameter specifies the number of additional points to draw from the current distribution before selecting the best. If a negative value is specified, a heuristic is used to sample `O(log(num_to_sample))` additional points. Used only if `initial_clusters=KMeansClustering.KMEANS_PLUS_PLUS_INIT`.
- `relative_tolerance`: A relative tolerance of change in the loss between iterations. Stops learning if the loss changes less than this amount. This may not work correctly if `use_mini_batch=True`.
- `config`: See `tf.estimator.Estimator`.

Raises:

- `ValueError`: An invalid argument was passed to `initial_clusters` or `distance_metric`.

`cluster_centers`

```
cluster_centers()
```

Returns the cluster centers.

`evaluate`

```
evaluate(
    input_fn,
    steps=None,
    hooks=None,
    checkpoint_path=None,
    name=None
)
```

Evaluates the model given evaluation data input_fn.

For each step, calls `input_fn`, which returns one batch of data. Evaluates until: - `steps` batches are processed, or - `input_fn` raises an end-of-input exception (`OutOfRangeError` or `StopIteration`).

Args:

- `input_fn`: Input function returning a tuple of: features - Dictionary of string feature name to `Tensor` or `SparseTensor`. labels - `Tensor` or dictionary of `Tensor` with labels.
- `steps`: Number of steps for which to evaluate model. If `None`, evaluates until `input_fn` raises an end-of-input exception.
- `hooks`: List of `SessionRunHook` subclass instances. Used for callbacks inside the evaluation call.
- `checkpoint_path`: Path of a specific checkpoint to evaluate. If `None`, the latest checkpoint in `model_dir` is used.
- `name`: Name of the evaluation if user needs to run multiple evaluations on different data sets, such as on training data vs test data. Metrics for different evaluations are saved in separate folders, and appear separately in tensorboard.

Returns:

A dict containing the evaluation metrics specified in `model_fn` keyed by name, as well as an entry `global_step` which contains the value of the global step for which this evaluation was performed.

Raises:

- `ValueError`: If `steps <= 0`.
- `ValueError`: If no model has been trained, namely `model_dir`, or the given `checkpoint_path` is empty.

`export_savedmodel`

```
export_savedmodel(
    export_dir_base,
    serving_input_receiver_fn,
    assets_extra=None,
    as_text=False,
    checkpoint_path=None
)
```

Exports inference graph as a SavedModel into given dir.

This method builds a new graph by first calling the serving_input_receiver_fn to obtain feature `Tensor`s, and then calling this `Estimator`'s model_fn to generate the model graph based on those features. It restores the given checkpoint (or, lacking that, the most recent checkpoint) into this graph in a fresh session. Finally it creates a timestamped export directory below the given export_dir_base, and writes a `SavedModel` into it containing a single `MetaGraphDef` saved from this session.

The exported `MetaGraphDef` will provide one `SignatureDef` for each element of the export_outputs dict returned from the model_fn, named using the same keys. One of these keys is always signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY, indicating which signature will be served when a serving request does not specify one. For each signature, the outputs are provided by the corresponding `ExportOutput`s, and the inputs are always the input receivers provided by the serving_input_receiver_fn.

Extra assets may be written into the SavedModel via the extra_assets argument. This should be a dict, where each key gives a destination path (including the filename) relative to the assets.extra directory. The corresponding value gives the full path of the source file to be copied. For example, the simple case of copying a single file without renaming it is specified as `{'my_asset_file.txt': '/path/to/my_asset_file.txt'}`.

Args:

- `export_dir_base`: A string containing a directory in which to create timestamped subdirectories containing exported SavedModels.
- `serving_input_receiver_fn`: A function that takes no argument and returns a `ServingInputReceiver`.
- `assets_extra`: A dict specifying how to populate the assets.extra directory within the exported SavedModel, or `None` if no extra assets are needed.
- `as_text`: whether to write the SavedModel proto in text format.
- `checkpoint_path`: The checkpoint path to export. If `None` (the default), the most recent checkpoint found within the model directory is chosen.

Returns:

The string path to the exported directory.

Raises:

- `ValueError`: if no serving_input_receiver_fn is provided, no export_outputs are provided, or no checkpoint can be found.

`get_variable_names`

```
get_variable_names()
```

Returns list of all variable names in this model.

Returns:

List of names.

Raises:

- `ValueError` : If the Estimator has not produced a checkpoint yet.

`get_variable_value`

```
get_variable_value(name)
```

Returns value of the variable given by name.

Args:

- `name` : string or a list of string, name of the tensor.

Returns:

Numpy array - value of the tensor.

Raises:

- `ValueError` : If the Estimator has not produced a checkpoint yet.

`latest_checkpoint`

```
latest_checkpoint()
```

Finds the filename of latest saved checkpoint file in **`model_dir`** .

Returns:

The full path to the latest checkpoint or **`None`** if no checkpoint was found.

`predict`

```
predict(
    input_fn,
    predict_keys=None,
    hooks=None,
    checkpoint_path=None
)
```

Yields predictions for given features.

Args:

- `input_fn`: Input function returning features which is a dictionary of string feature name to `Tensor` or `SparseTensor`. If it returns a tuple, first item is extracted as features. Prediction continues until `input_fn` raises an end-of-input exception (`OutOfRangeError` or `StopIteration`).
- `predict_keys`: list of `str`, name of the keys to predict. It is used if the `EstimatorSpec.predictions` is a `dict`. If `predict_keys` is used then rest of the predictions will be filtered from the dictionary. If `None`, returns all.
- `hooks`: List of `SessionRunHook` subclass instances. Used for callbacks inside the prediction call.
- `checkpoint_path`: Path of a specific checkpoint to predict. If `None`, the latest checkpoint in `model_dir` is used.

Yields:

Evaluated values of `predictions` tensors.

Raises:

- `ValueError`: Could not find a trained model in model_dir.
- `ValueError`: if batch length of predictions are not same.
- `ValueError`: If there is a conflict between `predict_keys` and `predictions`. For example if `predict_keys` is not `None` but `EstimatorSpec.predictions` is not a `dict`.

`predict_cluster_index`

```
predict_cluster_index(input_fn)
```

Finds the index of the closest cluster center to each input point.

Args:

- `input_fn`: Input points. See `tf.estimator.Estimator.predict`.

Yields:

The index of the closest cluster center for each input point.

`score`

```
score(input_fn)
```

Returns the sum of squared distances to nearest clusters.

Note that this function is different from the corresponding one in sklearn which returns the negative sum.

Args:

- `input_fn`: Input points. See `tf.estimator.Estimator.evaluate`. Only one batch is retrieved.

Returns:

The sum of the squared distance from each point in the first batch of inputs to its nearest cluster center.

`train`

```
train(
    input_fn,
    hooks=None,
    steps=None,
    max_steps=None,
    saving_listeners=None
)
```

Trains a model given training data input_fn.

Args:

- `input_fn`: Input function returning a tuple of: features - `Tensor` or dictionary of string feature name to `Tensor`. labels - `Tensor` or dictionary of `Tensor` with labels.
- `hooks`: List of `SessionRunHook` subclass instances. Used for callbacks inside the training loop.
- `steps`: Number of steps for which to train model. If `None`, train forever or train until input_fn generates the `OutOfRange` error or `StopIteration` exception. 'steps' works incrementally. If you call two times train(steps=10) then training occurs in total 20 steps. If `OutOfRange` or `StopIteration` occurs in the middle, training stops before 20 steps. If you don't want to have incremental behavior please set `max_steps` instead. If set, `max_steps` must be `None`.
- `max_steps`: Number of total steps for which to train model. If `None`, train forever or train until input_fn generates the `OutOfRange` error or `StopIteration` exception. If set, `steps` must be `None`. If `OutOfRange` or `StopIteration` occurs in the middle, training stops before `max_steps` steps. Two calls to `train(steps=100)` means 200 training iterations. On the other hand, two calls to `train(max_steps=100)` means that the second call will not do any iteration since first call did all 100 steps.
- `saving_listeners`: list of `CheckpointSaverListener` objects. Used for callbacks that run immediately before or after checkpoint savings.

Returns:

`self`, for chaining.

Raises:

- `ValueError`: If both `steps` and `max_steps` are not `None`.
- `ValueError`: If either `steps` or `max_steps` is <= 0.

`transform`

```
transform(input_fn)
```

Transforms each input point to its distances to all cluster centers.

Note that if `distance_metric=KMeansClustering.SQUARED_EUCLIDEAN_DISTANCE`, this function returns the squared Euclidean distance while the corresponding sklearn function returns the Euclidean distance.

Args:

- `input_fn`: Input points. See `tf.estimator.Estimator.predict`.

Yields:

The distances from each input point to each cluster center.

Class Members

`ALL_DISTANCES`

`CLUSTERS`

`CLUSTER_INDEX`

`COSINE_DISTANCE`

`KMEANS_PLUS_PLUS_INIT`

`RANDOM_INIT`

`SCORE`

`SQUARED_EUCLIDEAN_DISTANCE`