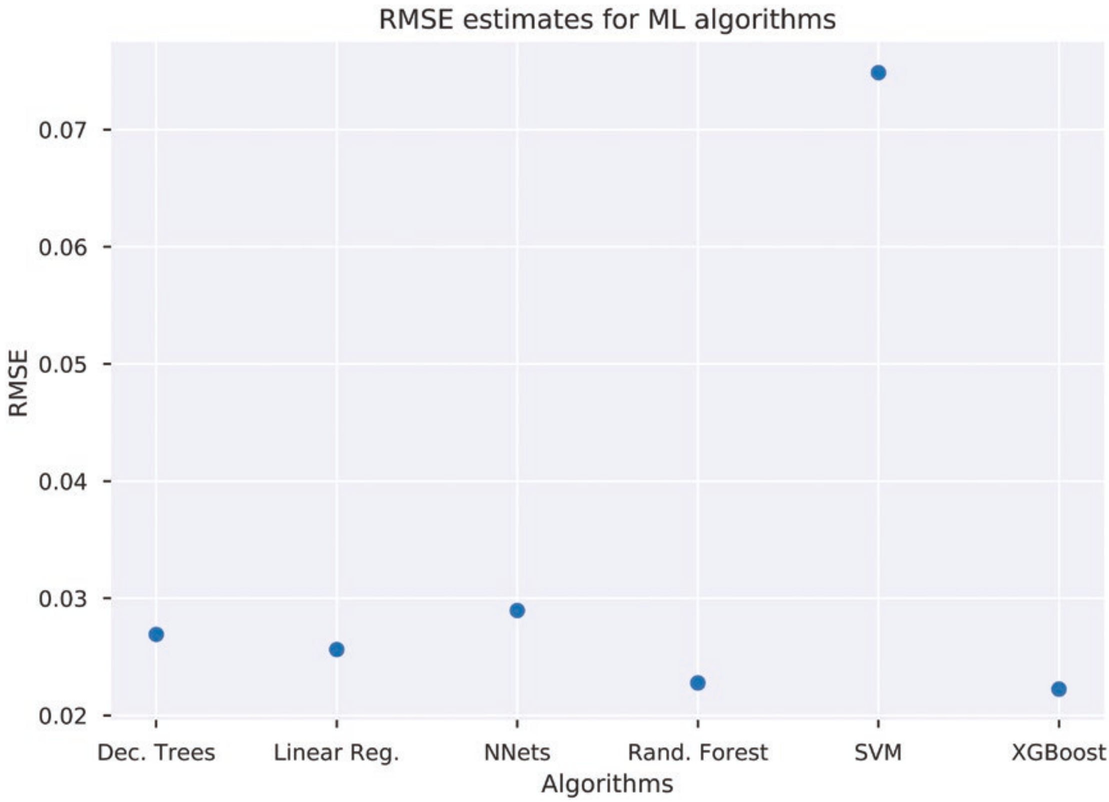

Figure 44-3. *RMSE estimates for ML algorithms*

Dataflow and TensorFlow Transform for Large-Scale Data Processing

In this section, we use Google Cloud Dataflow to carry out large-scale data processing on humongous datasets. Google Dataflow as earlier discussed is a serverless, parallel, and distributed infrastructure for running jobs for batch and stream data processing. Dataflow is a vital component in architecting a production pipeline for building and deploying large-scale machine learning products. In conjunction with Cloud Dataflow, we use TensorFlow Transform (TFT), a library built for preprocessing with Tensorflow. The goal of using TFT is to have a consistent set of transformation operations applied to the dataset when the model is trained and when it is served or deployed for consumption. In the following steps, each code block is executed in a Notebook cell:

- Import the relevant libraries. Remember that Apache Beam (as of now) only supports Python 2. Moreso, TFT only works with a specific combination of Tensorflow and Apache Beam packages. In this case, TFT 0.8.0 works with TF 1.8 and Apache Beam [GCP] 2.5.0. After importing the libraries, be sure to **restart the Notebook kernel**.

  At this point, change the Notebook runtime type to Python 2.0.

```
%%bash
source activate py2env
pip install --upgrade tensorflow
pip install --upgrade apache-beam[gcp]
pip install --upgrade tensorflow_transform==0.8.0
apt-get install libsnappy-dev
pip install --upgrade python-snappy==0.5.1
```

  Restart the kernel after you do a pip install.

- Connect to GCP.

```
from google.colab import auth
auth.authenticate_user()
print('Authenticated')

# configure GCP project - update with your parameters
project_id = 'ekabasandbox'
bucket_name = 'superconductor'
region = 'us-central1'
tf_version = '1.8'

# configure gcloud
!gcloud config set project {project_id}
!gcloud config set compute/region {region}
```

- Create query method for retrieving training and testing datasets from BigQuery.

```
def create_query(phase, EVERY_N=None):
    """
```

```
EVERY_N: Integer. Sample one out of every N rows from the full
dataset. Larger values will yield smaller sample
phase: 1=train 2=valid
"""
base_query = """
WITH super_df AS (
  SELECT
    number_of_elements, mean_atomic_mass, wtd_mean_atomic_mass,
    gmean_atomic_mass, wtd_gmean_atomic_mass, entropy_atomic_
    mass,
    wtd_entropy_atomic_mass, range_atomic_mass, wtd_range_
    atomic_mass,
    std_atomic_mass, wtd_std_atomic_mass, mean_fie, wtd_mean_fie,
    gmean_fie, wtd_gmean_fie, entropy_fie, wtd_entropy_fie,
    range_fie,
    wtd_range_fie, std_fie, wtd_std_fie, mean_atomic_radius,
    wtd_mean_atomic_radius,
    gmean_atomic_radius, wtd_gmean_atomic_radius, entropy_
    atomic_radius,
    wtd_entropy_atomic_radius, range_atomic_radius, wtd_range_
    atomic_radius,
    std_atomic_radius, wtd_std_atomic_radius, mean_Density,
    wtd_mean_Density,
    gmean_Density, wtd_gmean_Density, entropy_Density, wtd_
    entropy_Density,
    range_Density, wtd_range_Density, std_Density, wtd_std_
    Density, mean_ElectronAffinity,
    wtd_mean_ElectronAffinity, gmean_ElectronAffinity, wtd_
    gmean_ElectronAffinity
    entropy_ElectronAffinity, wtd_entropy_ElectronAffinity,
    range_ElectronAffinity,
    wtd_range_ElectronAffinity, std_ElectronAffinity, wtd_std_
    ElectronAffinity,
    mean_FusionHeat, wtd_mean_FusionHeat, gmean_FusionHeat,
    wtd_gmean_FusionHeat,
```

```
      entropy_FusionHeat, wtd_entropy_FusionHeat, range_FusionHeat,
      wtd_range_FusionHeat, std_FusionHeat, wtd_std_FusionHeat,
      mean_ThermalConductivity,
      wtd_mean_ThermalConductivity, gmean_ThermalConductivity,
      wtd_gmean_ThermalConductivity,
      entropy_ThermalConductivity, wtd_entropy_
      ThermalConductivity, range_ThermalConductivity,
      wtd_range_ThermalConductivity, std_ThermalConductivity,
      wtd_std_ThermalConductivity,
      mean_Valence, wtd_mean_Valence, gmean_Valence, wtd_gmean_
      Valence,
      entropy_Valence, wtd_entropy_Valence, range_Valence, wtd_
      range_Valence,
      std_Valence, wtd_std_Valence, critical_temp, ROW_NUMBER()
      OVER (PARTITION BY number_of_elements) row_num
    FROM
      `superconductor.superconductor`)

    SELECT
      *
    FROM
      super_df
"""

if EVERY_N == None:
    if phase < 2:
        # training
        query = "{0} WHERE MOD(row_num,4) < 2".format(base_
        query)
    else:
        query = "{0} WHERE MOD(row_num,4) = {1}".format(base_
        query, phase)
else:
    query = "{0} WHERE MOD(row_num,{1}) = {2}".format(base_
    query, EVERY_N, phase)

return query
```

- Create requirements.txt file to install dependencies (in this case tensorflow_transform) on Dataflow worker machines.

```
%%writefile requirements.txt
tensorflow-transform==0.8.0
```

- The following code block uses Apache Beam to build a data preprocessing pipeline to transform the raw dataset into a form suitable for building a predictive model. The transformation is the same procedure as done earlier with the reduced dataset, which included removing columns that had a high correlation and scaling the dataset numeric values to be within the same range. The output of the preprocessing pipeline produces a training set and an evaluation set. The Beam pipeline also uses TensorFlow Transform to save the metadata (both raw and processed) of the data transformation, as well as the transformed graph which can later be used as part of the serving function of the deployed model. We made this example to include the use of TensorFlow Transform for reference purposes.

```
import datetime
import snappy
import tensorflow as tf
import apache_beam as beam
import tensorflow_transform as tft
from tensorflow_transform.beam import impl as beam_impl

def get_table_header(projection_fields):
    header = ''
    for cnt, val in enumerate(projection_fields):
        if cnt > 0:
            header+=','+val
        else:
            header+=val
    return header

def preprocess_tft(inputs):
    result = {}
```

```
    for attr, value in inputs.items():
        result[attr] = tft.scale_to_0_1(value)

    return result

def cleanup(rowdict):
    # pull columns from BQ and create a line
    CSV_COLUMNS = 'number_of_elements,mean_atomic_mass,entropy_
    atomic_mass,wtd_entropy_atomic_mass,range_atomic_mass,wtd_
    range_atomic_mass,mean_fie,wtd_mean_fie,wtd_entropy_
    fie,range_fie,wtd_range_fie,mean_atomic_radius,wtd_mean_
    atomic_radius,range_atomic_radius,wtd_range_atomic_
    radius,mean_Density,entropy_Density,wtd_entropy_Density,range_
    Density,wtd_range_Density,mean_ElectronAffinity,wtd_
    entropy_ElectronAffinity,range_ElectronAffinity,wtd_range_
    ElectronAffinity,mean_FusionHeat,gmean_FusionHeat,entropy_
    FusionHeat,wtd_entropy_FusionHeat,range_FusionHeat,wtd_
    range_FusionHeat,mean_ThermalConductivity,wtd_mean_
    ThermalConductivity,gmean_ThermalConductivity,entropy_
    ThermalConductivity,wtd_entropy_ThermalConductivity,
    range_ThermalConductivity,wtd_range_ThermalConductivity,
    mean_Valence,wtd_mean_Valence,range_Valence,wtd_range_
    Valence,wtd_std_Valence,critical_temp'.split(',')

    def tofloat(value, ifnot):
        try:
            return float(value)
        except (ValueError, TypeError):
            return ifnot

    result = {
      k : tofloat(rowdict[k], -99) if k in rowdict else -99 for k
      in CSV_COLUMNS
    }

    row = ('{}'+',{}'*(len(result)-1)).format(result['number_of_
    elements'],result['mean_atomic_mass'],
```

```
            result['entropy_atomic_mass'], result['wtd_entropy_atomic_
            mass'],result['range_atomic_mass'],
            result['wtd_range_atomic_mass'],result['mean_fie'],
            result['wtd_mean_fie'],
            result['wtd_entropy_fie'],result['range_fie'],result['wtd_
            range_fie'],
            result['mean_atomic_radius'],result['wtd_mean_atomic_radius'],
            result['range_atomic_radius'],result['wtd_range_atomic_
            radius'],result['mean_Density'],
            result['entropy_Density'],result['wtd_entropy_Density'],
            result['range_Density'],
            result['wtd_range_Density'],result['mean_ElectronAffinity'],
            result['wtd_entropy_ElectronAffinity'],result['range_
            ElectronAffinity'],
            result['wtd_range_ElectronAffinity'],result['mean_
            FusionHeat'],result['gmean_FusionHeat'],
            result['entropy_FusionHeat'],result['wtd_entropy_
            FusionHeat'],result['range_FusionHeat'],
            result['wtd_range_FusionHeat'],result['mean_
            ThermalConductivity'],
            result['wtd_mean_ThermalConductivity'],result['gmean_
            ThermalConductivity'],
            result['entropy_ThermalConductivity'],result['wtd_entropy_
            ThermalConductivity'],
            result['range_ThermalConductivity'],result['wtd_range_
            ThermalConductivity'],
            result['mean_Valence'],result['wtd_mean_Valence'],
            result['range_Valence'],
            result['wtd_range_Valence'],result['wtd_std_Valence'],
            result['critical_temp'])
        yield row

    def preprocess():
        import os
        import os.path
        import datetime
```

```
from apache_beam.io import WriteToText
from apache_beam.io import tfrecordio
from tensorflow_transform.coders import example_proto_coder
from tensorflow_transform.tf_metadata import dataset_metadata
from tensorflow_transform.tf_metadata import dataset_schema
from tensorflow_transform.beam import tft_beam_io
from tensorflow_transform.beam.tft_beam_io import transform_
fn_io

job_name = 'preprocess-features' + '-' + datetime.datetime.
now().strftime('%y%m%d-%H%M%S')

print 'Launching Dataflow job {} ... hang on'.format(job_name)
OUTPUT_DIR = 'gs://{0}/preproc_csv/'.format(bucket_name)
import subprocess
subprocess.call('gsutil rm -r {}'.format(OUTPUT_DIR).split())
EVERY_N = 3

options = {
  'staging_location': os.path.join(OUTPUT_DIR, 'tmp', 'staging'),
  'temp_location': os.path.join(OUTPUT_DIR, 'tmp'),
  'job_name': job_name,
  'project': project_id,
  'max_num_workers': 24,
  'teardown_policy': 'TEARDOWN_ALWAYS',
  'no_save_main_session': True,
  'requirements_file': 'requirements.txt'
}
opts = beam.pipeline.PipelineOptions(flags=[], **options)
RUNNER = 'DataflowRunner'

# set up metadata
raw_data_schema = {
  colname : dataset_schema.ColumnSchema(tf.float32, [],
  dataset_schema.FixedColumnRepresentation())
                  for colname in 'number_of_elements,mean_atomic_
                  mass,entropy_atomic_mass,wtd_entropy_atomic_
                  mass,range_atomic_mass,wtd_range_atomic_mass,
```

```
                                   mean_fie,wtd_mean_fie,wtd_entropy_fie,range_
                                   fie,wtd_range_fie,mean_atomic_radius,wtd_
                                   mean_atomic_radius,range_atomic_radius,wtd_
                                   range_atomic_radius,mean_Density,entropy_
                                   Density,wtd_entropy_Density,range_Density,
                                   wtd_range_Density,mean_ElectronAffinity,wtd_
                                   entropy_ElectronAffinity,range_
                                   ElectronAffinity,wtd_range_ElectronAffinity,
                                   mean_FusionHeat,gmean_FusionHeat,entropy_
                                   FusionHeat,wtd_entropy_FusionHeat,range_
                                   FusionHeat,wtd_range_FusionHeat,
                                   mean_ThermalConductivity,
                                   wtd_mean_ThermalConductivity,
                                   gmean_ThermalConductivity,
                                   entropy_ThermalConductivity,
                                   wtd_entropy_ThermalConductivity,
                                   range_ThermalConductivity,wtd_range_
                                   ThermalConductivity,mean_Valence,wtd_mean_
                                   Valence,range_Valence,wtd_range_Valence,wtd_
                                   std_Valence,critical_temp'.split(',')
}
raw_data_metadata = dataset_metadata.DatasetMetadata(dataset_
schema.Schema(raw_data_schema))

# run Beam
with beam.Pipeline(RUNNER, options=opts) as p:
    with beam_impl.Context(temp_dir=os.path.join
    (OUTPUT_DIR, 'tmp')):
        # save the raw data metadata
        _ = (raw_data_metadata
          | 'WriteInputMetadata' >> tft_beam_io.WriteMetadata(
              os.path.join(OUTPUT_DIR, 'metadata/rawdata_
              metadata'),
              pipeline=p))

        projection_fields = ['number_of_elements',
        'mean_atomic_mass', 'entropy_atomic_mass',
```

```
'wtd_entropy_atomic_mass',
'range_atomic_mass',
'wtd_range_atomic_mass', 'mean_
fie', 'wtd_mean_fie',
'wtd_entropy_fie', 'range_fie',
'wtd_range_fie',
'mean_atomic_radius', 'wtd_mean_
atomic_radius',
'range_atomic_radius', 'wtd_
range_atomic_radius', 'mean_
Density',
'entropy_Density', 'wtd_entropy_
Density', 'range_Density',
'wtd_range_Density', 'mean_
ElectronAffinity',
'wtd_entropy_ElectronAffinity',
'range_ElectronAffinity',
'wtd_range_ElectronAffinity',
'mean_FusionHeat', 'gmean_
FusionHeat',
'entropy_FusionHeat', 'wtd_
entropy_FusionHeat', 'range_
FusionHeat',
'wtd_range_FusionHeat', 'mean_
ThermalConductivity',
'wtd_mean_ThermalConductivity',
'gmean_ThermalConductivity',
'entropy_ThermalConductivity',
'wtd_entropy_
ThermalConductivity',
'range_ThermalConductivity',
'wtd_range_ThermalConductivity',
'mean_Valence', 'wtd_mean_
Valence', 'range_Valence',
'wtd_range_Valence', 'wtd_std_
Valence', 'critical_temp']
```

```
header = get_table_header(projection_fields)

# analyze and transform training
raw_data = (p
  | 'train_read' >> beam.io.Read(beam.
  io.BigQuerySource(query=create_query(1, EVERY_N),
  use_standard_sql=True)))

raw_dataset = (raw_data, raw_data_metadata)
transformed_dataset, transform_fn = (
    raw_dataset | beam_impl.AnalyzeAndTransformDataset
    (preprocess_tft))
transformed_data, transformed_metadata = transformed_
dataset

_ = (transformed_data
     | 'train_filter' >> beam.FlatMap(cleanup)
     | 'WriteTrainData' >> beam.io.Write(beam.
       io.WriteToText(
          file_path_prefix=os.path.join(OUTPUT_DIR,
          'data', 'train'),
          file_name_suffix=".csv",
          shard_name_template="-SS-of-NN",
          header=header,
          num_shards=1)))

# transform eval data
raw_test_data = (p
  | 'eval_read' >> beam.io.Read(beam.
    io.BigQuerySource(query=create_query(2, EVERY_N),
    use_standard_sql=True)))

raw_test_dataset = (raw_test_data, raw_data_metadata)
transformed_test_dataset = (
    (raw_test_dataset, transform_fn) | beam_impl.
    TransformDataset())
transformed_test_data, _ = transformed_test_dataset
```

```
        _ = (transformed_test_data
             | 'eval_filter' >> beam.FlatMap(cleanup)
             | 'WriteTestData' >> beam.io.Write(beam.
               io.WriteToText(
                    file_path_prefix=os.path.join(OUTPUT_DIR,
                    'data', 'eval'),
                    file_name_suffix=".csv",
                    shard_name_template="-SS-of-NN",
                    num_shards=1)))

        _ = (transform_fn
             | 'WriteTransformFn' >>
             transform_fn_io.WriteTransformFn(os.path.
             join(OUTPUT_DIR, 'metadata')))

preprocess()
```

- The Dataflow pipeline graph is shown in Figure 44-4.

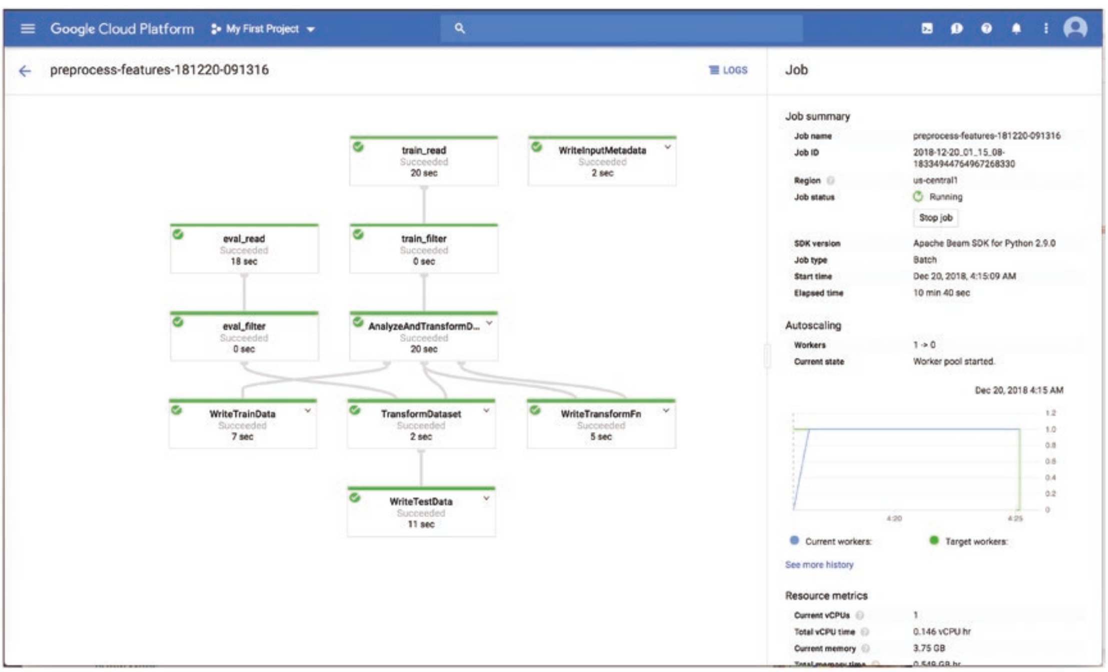

__Figure 44-4.__ Dataflow pipeline graph

Training on Cloud MLE

The following code example will train the processed datasets on Google Cloud MLE. At this point, change the Notebook runtime type to Python 3.0.

- Configure GCP project.

```
# configure GCP project - update with your parameters
project_id = 'ekabasandbox'
bucket_name = 'superconductor'
region = 'us-central1'
tf_version = '1.8'

import os
os.environ['bucket_name'] = bucket_name
os.environ['tf_version'] = tf_version
os.environ['project_id'] = project_id
os.environ['region'] = region
```

- Create directory "trainer".

```
# create directory trainer
import os
try:
    os.makedirs('./trainer')
    print('directory created')
except OSError:
    print('could not create directory')
```

- Create file __init__.py.

```
%%writefile trainer/__init__.py
```

- Create the trainer file task.py. Replace the bucket name with your values.

```
%%writefile trainer/task.py
import argparse
import json
import os
```