```
import pickle
import numpy
import math

from sklearn.preprocessing import StandardScaler

from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Merge, Reshape
from keras.layers.embeddings import Embedding
from keras.callbacks import ModelCheckpoint

from prepare_nn_features import split_features


class Model(object):

    def __init__(self, train_ratio):
        self.train_ratio = train_ratio
        self.__load_data()

    def evaluate(self):
        if self.train_ratio == 1:
            return 0
        total_sqe = 0
        num_real_test = 0
        for record, sales in zip(self.X_val, self.y_val):
            if sales == 0:
                continue
            guessed_sales = self.guess(record)
            sqe = ((sales - guessed_sales) / sales) ** 2
            total_sqe += sqe
            num_real_test += 1
        result = math.sqrt(total_sqe / num_real_test)
        return result

    def __load_data(self):
        f = open('feature_train_data.pickle', 'rb')
        (self.X, self.y) = pickle.load(f)
        self.X = numpy.array(self.X)
        self.y = numpy.array(self.y)
        self.num_records = len(self.X)
        self.train_size = int(self.train_ratio * self.num_records)
        self.test_size = self.num_records - self.train_size
        self.X, self.X_val = self.X[:self.train_size], self.X[self.train_
size:]
        self.y, self.y_val = self.y[:self.train_size], self.y[self.train_
size:]


class NN_with_EntityEmbedding(Model):
  def __init__(self, train_ratio):
    super().__init__(train_ratio)
    self.build_preprocessor(self.X)
    self.nb_epoch = 20
    self.checkpointer = ModelCheckpoint(filepath="best_model_weights.
hdf5", verbose=1, save_best_only=True)
    self.max_log_y = numpy.max(numpy.log(self.y))
    self.min_log_y = numpy.min(numpy.log(self.y))
    self.__build_keras_model()
    self.fit()

  def build_preprocessor(self, X):
    X_list = split_features(X)
    # Google trend de
    self.gt_de_enc = StandardScaler()
    self.gt_de_enc.fit(X_list[32])
    # Google trend state
    self.gt_state_enc = StandardScaler()
    self.gt_state_enc.fit(X_list[33])

  def preprocessing(self, X):
    X_list = split_features(X)
    X_list[32] = self.gt_de_enc.transform(X_list[32])
    X_list[33] = self.gt_state_enc.transform(X_list[33])
    return X_list

  def __build_keras_model(self):
    models = []

    model_store = Sequential()
    model_store.add(Embedding(1115, 50, input_length=1))
    model_store.add(Reshape(dims=(50,)))
    models.append(model_store)

    model_dow = Sequential()
    model_dow.add(Embedding(7, 6, input_length=1))
    model_dow.add(Reshape(dims=(6,)))
    models.append(model_dow)

    model_promo = Sequential()
    model_promo.add(Dense(1, input_dim=1))
    models.append(model_promo)

    model_year = Sequential()
    model_year.add(Embedding(3, 2, input_length=1))
    model_year.add(Reshape(dims=(2,)))
    models.append(model_year)

    model_month = Sequential()
    model_month.add(Embedding(12, 6, input_length=1))
    model_month.add(Reshape(dims=(6,)))
```

```
models.append(model_month)

model_day = Sequential()
model_day.add(Embedding(31, 10, input_length=1))
model_day.add(Reshape(dims=(10,)))
models.append(model_day)

model_stateholiday = Sequential()
model_stateholiday.add(Embedding(4, 3, input_length=1))
model_stateholiday.add(Reshape(dims=(3,)))
models.append(model_stateholiday)

model_school = Sequential()
model_school.add(Dense(1, input_dim=1))
models.append(model_school)

model_competemonths = Sequential()
model_competemonths.add(Embedding(25, 2, input_length=1))
model_competemonths.add(Reshape(dims=(2,)))
models.append(model_competemonths)

model_promo2weeks = Sequential()
model_promo2weeks.add(Embedding(26, 1, input_length=1))
model_promo2weeks.add(Reshape(dims=(1,)))
models.append(model_promo2weeks)

model_lastestpromo2months = Sequential()
model_lastestpromo2months.add(Embedding(4, 1, input_length=1))
model_lastestpromo2months.add(Reshape(dims=(1,)))
models.append(model_lastestpromo2months)

model_distance = Sequential()
model_distance.add(Dense(1, input_dim=1))
models.append(model_distance)

model_storetype = Sequential()
model_storetype.add(Embedding(5, 2, input_length=1))
model_storetype.add(Reshape(dims=(2,)))
models.append(model_storetype)

model_assortment = Sequential()
model_assortment.add(Embedding(4, 3, input_length=1))
model_assortment.add(Reshape(dims=(3,)))
models.append(model_assortment)

model_promointerval = Sequential()
model_promointerval.add(Embedding(4, 3, input_length=1))
model_promointerval.add(Reshape(dims=(3,)))
models.append(model_promointerval)

model_competyear = Sequential()
model_competyear.add(Embedding(18, 4, input_length=1))
model_competyear.add(Reshape(dims=(4,)))
models.append(model_competyear)

model_promotyear = Sequential()
model_promotyear.add(Embedding(8, 4, input_length=1))
model_promotyear.add(Reshape(dims=(4,)))
models.append(model_promotyear)

model_germanstate = Sequential()
model_germanstate.add(Embedding(12, 6, input_length=1))
model_germanstate.add(Reshape(dims=(6,)))
models.append(model_germanstate)

model_woy = Sequential()
model_woy.add(Embedding(53, 2, input_length=1))
model_woy.add(Reshape(dims=(2,)))
models.append(model_woy)

model_temperature = Sequential()
model_temperature.add(Dense(3, input_dim=3))
models.append(model_temperature)

model_humidity = Sequential()
model_humidity.add(Dense(3, input_dim=3))
models.append(model_humidity)

model_wind = Sequential()
model_wind.add(Dense(2, input_dim=2))
models.append(model_wind)

model_cloud = Sequential()
model_cloud.add(Dense(1, input_dim=1))
models.append(model_cloud)

model_weatherevent = Sequential()
model_weatherevent.add(Embedding(22, 4, input_length=1))
model_weatherevent.add(Reshape(dims=(4,)))
models.append(model_weatherevent)

model_promo_forward = Sequential()
model_promo_forward.add(Embedding(8, 1, input_length=1))
model_promo_forward.add(Reshape(dims=(1,)))
models.append(model_promo_forward)

model_promo_backward = Sequential()
model_promo_backward.add(Embedding(8, 1, input_length=1))
model_promo_backward.add(Reshape(dims=(1,)))
models.append(model_promo_backward)

model_stateholiday_forward = Sequential()
model_stateholiday_forward.add(Embedding(8, 1, input_length=1))
model_stateholiday_forward.add(Reshape(dims=(1,)))
```

```
        models.append(model_stateholiday_forward)

        model_sateholiday_backward = Sequential()
        model_sateholiday_backward.add(Embedding(8, 1, input_length=1))
        model_sateholiday_backward.add(Reshape(dims=(1,)))
        models.append(model_sateholiday_backward)

        model_stateholiday_count_forward = Sequential()
        model_stateholiday_count_forward.add(Embedding(3, 1, input_length=1))
        model_stateholiday_count_forward.add(Reshape(dims=(1,)))
        models.append(model_stateholiday_count_forward)

        model_stateholiday_count_backward = Sequential()
        model_stateholiday_count_backward.add(Embedding(3, 1, input_length=1))
        model_stateholiday_count_backward.add(Reshape(dims=(1,)))
        models.append(model_stateholiday_count_backward)

        model_schoolholiday_forward = Sequential()
        model_schoolholiday_forward.add(Embedding(8, 1, input_length=1))
        model_schoolholiday_forward.add(Reshape(dims=(1,)))
        models.append(model_schoolholiday_forward)

        model_schoolholiday_backward = Sequential()
        model_schoolholiday_backward.add(Embedding(8, 1, input_length=1))
        model_schoolholiday_backward.add(Reshape(dims=(1,)))
        models.append(model_schoolholiday_backward)

        model_googletrend_de = Sequential()
        model_googletrend_de.add(Dense(1, input_dim=1))
        models.append(model_googletrend_de)

        model_googletrend_state = Sequential()
        model_googletrend_state.add(Dense(1, input_dim=1))
        models.append(model_googletrend_state)

        # model_weather = Sequential()
        # model_weather.add(Merge([model_temperature, model_humidity, model_wind, model_weatherevent], mode='concat'))
        # model_weather.add(Dense(1))
        # model_weather.add(Activation('relu'))
        # models.append(model_weather)

        self.model = Sequential()
        self.model.add(Merge(models, mode='concat'))
        self.model.add(Dropout(0.02))
        self.model.add(Dense(1000, init='uniform'))
        self.model.add(Activation('relu'))
        self.model.add(Dense(500, init='uniform'))
        self.model.add(Activation('relu'))
        self.model.add(Dense(1))
        self.model.add(Activation('sigmoid'))

        self.model.compile(loss='mean_absolute_error', optimizer='adam')

    def _val_for_fit(self, val):
        val = numpy.log(val) / self.max_log_y
        return val

    def _val_for_pred(self, val):
        return numpy.exp(val * self.max_log_y)

    def fit(self):
        if self.train_ratio < 1:
            self.model.fit(self.preprocessing(self.X), self._val_for_fit(self.y),
                           validation_data=(self.preprocessing(self.X_val), self._val_for_fit(self.y_val)),
                           nb_epoch=self.nb_epoch, batch_size=128,
                           # callbacks=[self.checkpointer],
                           )
            # self.model.load_weights('best_model_weights.hdf5')
            print("Result on validation data: ", self.evaluate())
        else:
            self.model.fit(self.preprocessing(self.X), self._val_for_fit(self.y),
                           nb_epoch=self.nb_epoch, batch_size=128)

    def guess(self, feature):
        feature = numpy.array(feature).reshape(1, -1)
        return self._val_for_pred(self.model.predict(self.preprocessing(feature)))[0][0]
```