Data Science Interview Assignment Solution

Questions:

Please answer the following questions.

1. How long did it take you to solve the problem?
   Answer: It takes me two days to fully build and modify the model to solve the problem

2. What software language and libraries did you use to solve the problem?
   Answer: The language I used is Python. The libraries : Ipython, matplotlib, numpy, pandas, sklearn, tensorflow.
   Code are listed blew if interested:

```
from __future__ import print_function
import math
from IPython import display
from matplotlib import cm
from matplotlib import gridspec
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import os
import csv as csv
from sklearn import metrics
import tensorflow as tf
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import LabelEncoder
from tensorflow.python.data import Dataset
```

3. What steps did you take to prepare the data for the project? Was any cleaning necessary?

Answer: 1.The train_features data from the *train_ features_DATE.csv file* are merged with the train_salaries data from *train_salaries_DATE.csv* as the train_data on the same jobId. This is done for the purpose of relate the right features data with target for each sample.

2. The data with zero ‘salary’ values are removed from the training data since in reality no one should earn a salary with 0 value. Thus these data can be considered as the missing data. Keeping those data may mislead the model.

Also I preprocess the features before build the build. This will be discussed in question 6.

Code for import and preparing the data are listed blew if interested:

```
# import data and preprocessing the data
dataset_path = 'data'
train_datafile_features = os.path.join(dataset_path, 'train_features_2013-03-07.csv')
train_datafile_salaries = os.path.join(dataset_path, 'train_salaries_2013-03-07.csv')
test_datafile_features = os.path.join(dataset_path, 'train_features_2013-03-07.csv')
train_data_features = pd.read_csv(train_datafile_features)
train_data_salaries = pd.read_csv(train_datafile_salaries)
test_data_features = pd.read_csv(test_datafile_features)
train_data = pd.merge(train_data_features, train_data_salaries, on=['jobId'], how='left')
# remove the data with zero salary values
salary = list(train_data.salary)
index = salary.index(0)
while 0 in salary:
salary.remove(0)
train_data=train_data[train_data.salary.isin(salary)]
trainData_dataframe = train_data.reindex(
np.random.permutation(train_data.index))
```

4. What algorithmic method did you apply? Why? What other methods did you consider?

Answer: The algorithmic method I used is deep neural network(DNN). This algorithm is chosen based the following advantages of DNN:

(1)DNN have the ability to learn and model non-linear and complex relationships. Since no evidence has indicated that the relationship between features and salary would be linear, in fact the relationship between the features and the salary could be extremely complex, DNN might be an appropriate choice.
(2) Unlike many other prediction techniques, DNN does not impose any restrictions on the input variables to be normally distributed. Since no given features in the training data has shown a patterned distribution, the DNN model would be a safe choice.
I considered the linear regression model at first since it is the easiest to operate but quickly reject it since most of the features are regarded as categorical feature and no linear relation between the categorical feature and the target.

5. Describe how the algorithmic method that you chose works?

Answer: Deep neural network networks composed of several layers. The layers are made of *nodes*. A node is just a place where computation happens. A node combines input from the data with a set of weights, that either amplify or dampen that input, thereby assigning significance to inputs for the task the algorithm is trying to learn. These input-weight products are summed and the sum is passed through an activation function, to determine whether and to what extent that signal progresses further through the network to affect the ultimate outcome which is the prediction of salary in this case. Each layer's output is simultaneously the subsequent layer's input, starting from an initial input layer receiving the data. DNN improved its performance based on the backpropagation algorithm by adjusting the weight to minimize the loss function(gradient descent algorithm). Each node layer in a deep network learns features automatically by repeatedly trying to reconstruct the input from which it draws its samples, attempting to minimize the difference between the network's guesses and the probability distribution of the input data itself.
In my model, the DNN is composed with the input layer,3 hidden layers with 256,128,64 nodes and the output layer is the prediction of salary.

6. What features did you use? Why?

Answer: The features I used are: "companyId", "jobType","degree","major", "industry", "yearsExperience", "milesFromMetropolis".

The only feature not used is the "JobId" since it is the identification of each sample and is unique for each sample and will not contribute to the salary. Preprocess of features are conducted before input into the training model, among all the 7 features used in the model, the values of "companyId", "jobType", "degree", "major", "industry" are presented as string instead of integer or float type, by no means should we input the value of string type into the model, thus they are considered as categorical features and are transformed as categorical label such as 1 ,2,…,using LabelEncoder function. After transformation, the input values of those features will be labels.

Also the input values of "yearsExperience", "milesFromMetropolis" features are normalized to fall within the range (-1,1), this helps the SGD not get stuck taking steps that are two large in one dimension while too small in another.

Code for preprocess the features are listed blew if interested:

```
    def Labelencorder(data):
        encoder = LabelEncoder()
        companyId_label = encoder.fit_transform(data["companyId"])
        data["companyId"] = companyId_label
        jobType_label = encoder.fit_transform(data["jobType"])
        data["jobType"] = jobType_label
         degree_label = encoder.fit_transform(data["degree"])
        data["degree"] = degree_label
        major_label = encoder.fit_transform(data["major"])
        data["major"] = major_label
         industry_label = encoder.fit_transform(data["industry"])
         data["industry"] = industry_label
        return data
   trainData_dataframe = Labelencorder(trainData_dataframe)
  test_data_features = Labelencorder(test_data_features)
  test_examples = test_data_features
  def normalize_linear_scale(examples_dataframe):
          processed_features = examples_dataframe
        processed_features["yearsExperience"] =
        linear_scale(examples_dataframe["yearsExperience"])
        processed_features["milesFromMetropolis"]=linear_scale(examples_dataframe["
        milesFromMetropolis"])
         return processed_features
 normalized_dataframe =
normalize_linear_scale(preprocess_features(trainData_dataframe))
```

7. How did you train your model? During training, what issues concerned you?

Answer: The training examples data read from two train csv files are split as two dataset: training data and validation data where the size of training data is ¾ of the size of total training example data while the validation data is the rest ¼ of the training example. The model used is the *DNNRegressor* from the *tensorflow.estimator* library. The batch size is set to be 1000 which is the number of samples that going to be propagated through the network each time. As for the training data, each time after a batch of samples propagated through the network, the data is shuffled and the next batch of samples is extracted and input into the network. The process is keep doing until all training samples are processed into the network for one period. The totally model is trained for 14 periods and *tf.train.AdagradOptimizer* is chose as the optimizer with the learning rate set as 0.15. Dropout coefficient is set to be 0.25 to avoid the overfitting.

During the training, what concerns me is the possibility of overfitting and whether the model is trained efficiently(the setup of learning rate coefficient). Thus after each period the data is finished with training, the accuracy of trained model is evaluated both on training data and validation data, the prediction result is compared with the targets(actual salary value in the data set) based on the criteria of root mean square error(RMSE). For both training data and validation data, the RSME value after each period is stored into a list to compare with the RSME value from the previous period. Through comparison, it could be conclude whether the model is trained efficiently(the RMSE should decrease). Examining the difference between the RMSE between training data and validation data after each period guarantee the efficiency of the model, given the fact that the parameter of model could be adjust in time if overfitting appears or accuracy decease.

After the model is fully trained, the test example features are input into the model to get the predict result.

Code of training model function is listed below if interested, more explanation about code can be found in the code.py file.

```
def train_nn_regression_model(
    my_optimizer,
    steps,
    batch_size,
    hidden_units,
    training_examples,
    training_targets,
   validation_examples,
   validation_targets):
  periods = 14
  steps_per_period = steps / periods

  # Create a DNNRegressor object.
  my_optimizer = tf.contrib.estimator.clip_gradients_by_norm(my_optimizer, 5.0)
  dnn_regressor = tf.estimator.DNNRegressor(
    feature_columns=construct_feature_columns(training_examples),
    hidden_units=hidden_units,
    optimizer=my_optimizer,
    dropout = 0.25
  )

  # Create input functions.
  training_input_fn = lambda: my_input_fn(training_examples,
                        training_targets["salary"],
                        batch_size=batch_size)
  predict_training_input_fn = lambda: my_input_fn(training_examples,
                            training_targets["salary"],
                            num_epochs=1,
                            shuffle=False)
  predict_validation_input_fn = lambda: my_input_fn(validation_examples,
                             validation_targets["salary"],
                             num_epochs=1,
                             shuffle=False)

  print("Training model...")
  print("RMSE (on training data):")
  training_rmse = []
  validation_rmse = []
  for period in range (0, periods):
   # Train the model, starting from the prior state.
   dnn_regressor.train(
     input_fn=training_input_fn,
     steps=steps_per_period
   )
   # Take a break and compute predictions.
   training_predictions =
 dnn_regressor.predict(input_fn=predict_training_input_fn)
```

```
    training_predictions = np.array([item['predictions'][0] for item in
training_predictions])

    validation_predictions =
dnn_regressor.predict(input_fn=predict_validation_input_fn)
    validation_predictions = np.array([item['predictions'][0] for item in
validation_predictions])

    # Compute training and validation loss.
    training_root_mean_squared_error = math.sqrt(
        metrics.mean_squared_error(training_predictions, training_targets))
    validation_root_mean_squared_error = math.sqrt(
        metrics.mean_squared_error(validation_predictions, validation_targets))
    # Occasionally print the current loss.
    print("  period %02d : %0.2f" % (period, training_root_mean_squared_error))
    # Add the loss metrics from this period to our list.
    training_rmse.append(training_root_mean_squared_error)
    validation_rmse.append(validation_root_mean_squared_error)
  print("Model training finished.")

  # Output a graph of loss metrics over periods.
  plt.ylabel("RMSE")
  plt.xlabel("Periods")
  plt.title("Root Mean Squared Error vs. Periods")
  plt.tight_layout()
  plt.plot(training_rmse, label="training")
  plt.plot(validation_rmse, label="validation")
  plt.legend()
  plt.show()

  print("Final RMSE (on training data):   %0.2f" %
training_root_mean_squared_error)
  print("Final RMSE (on validation data): %0.2f" %
validation_root_mean_squared_error)

  return dnn_regressor
```

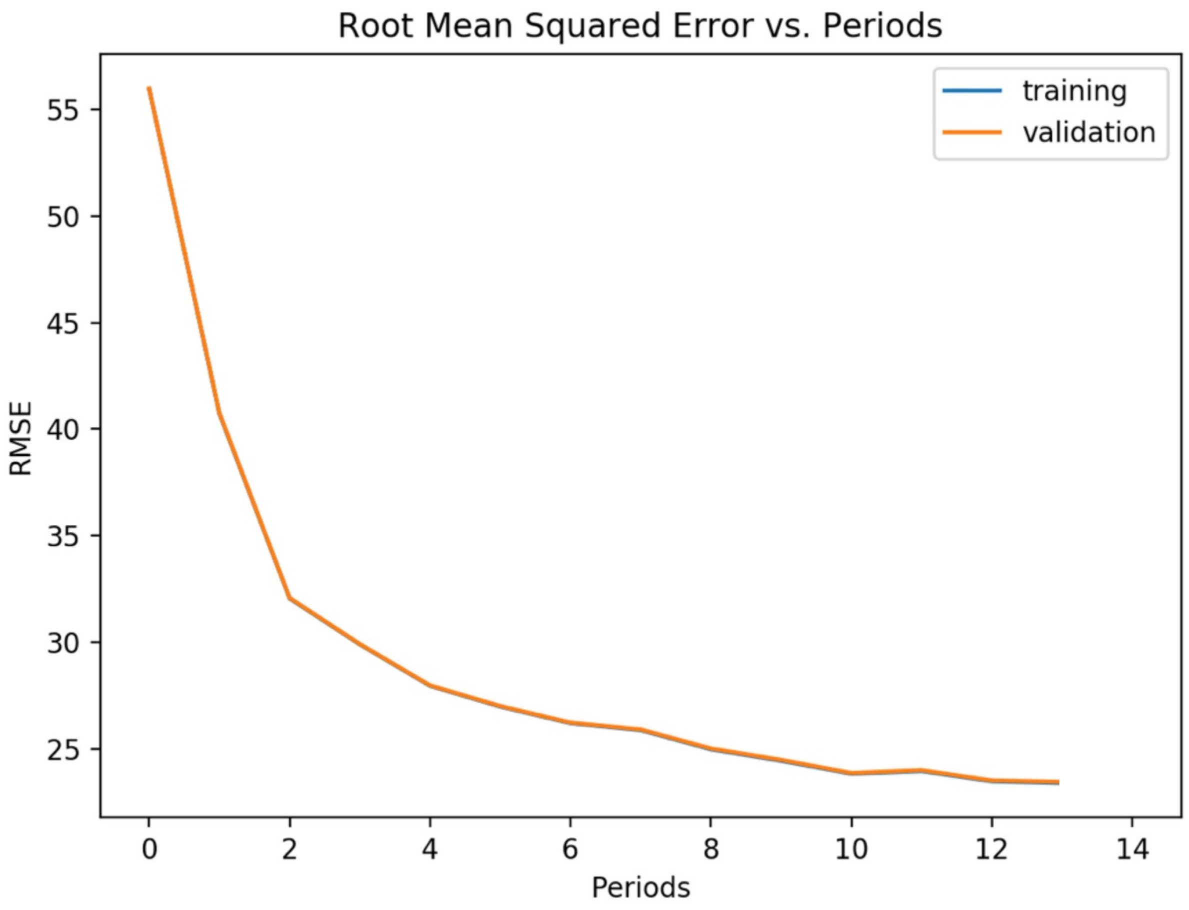
8. How did you assess the accuracy of your predictions? Why did you choose that method? Would you consider any alternative approaches for assessing accuracy?

Answer: Since no target data(salary) of test examples are provided, assess the accuracy could only be conducted on the training example data. Since the training example data are split as training dataset and validation dataset, the accuracy of prediction is assessed on both datasets through the comaprision of predition of salary using the trained model with the target(salary value in taring example data) by the critera of root mean square error(RMSE).

If both tarinig data and validation data have small RMSE value, the prediction will also be accurate on test examples.

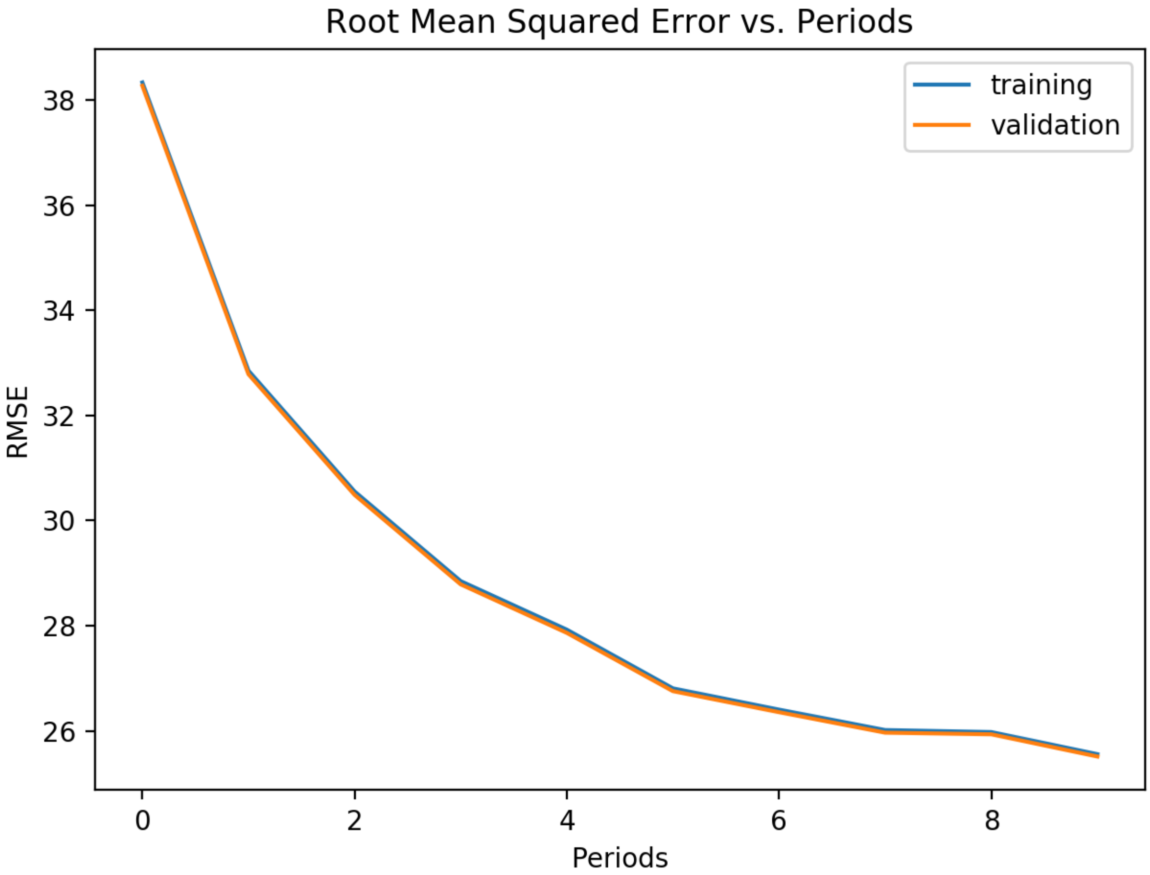
RMSE is an effective way to assess the accuracy since it is the statistic whose value is minimized during the parameter estimation process, and it is the statistic that determines the width of the confidence intervals for predictions. It is more sensitive than other measures to the occasional large error.

The figure blew shows the RSME plot for training data and validation data for final model :



9. Which features had the greatest impact on salary? How did you identify these to be most significant? Which features had the least impact on salary? How did you identify these?

Answer: The features had the greatest impact on salary are "jobType","degree","major", "industry", "yearsExperience". To prove this conclusion, another training on the same model was conducted with these five features as the input features, the blew figure shows RMSE on training and validation data for the first ten periods. It indicates that without the "companyId", "milesFromMetropolis", the prediction accuracy is not far from the prediction accuracy with all the features. The final RMSE for validation date with five input features ("jobType","degree","major", "industry", "yearsExperience") is 25.56 while the final RMSE for validation date with all features is 23.71. Thus "companyId", "milesFromMetropolis" has the least impact on salary.


Root Mean Squared Error vs. Periods
training
validation
RMSE
38
36
34
32
30
28
26
0
2
4
6
8
Periods