Sentiment Analysis Model Operation version 1.0.0

Beichen Su

8/29/2017

1. Introduction

This is the operation manual for the Sentiment Analysis Model which is create and will be maintained on the cloud. The work has been saved on "~/Sentiment_Model" under user rstudio, with password rstudio and can be accessed by "[redacted]".

The workflow of this model is given by: Load the data from SQL Server and clean the data, Tokenize the short sentences and create a document-term matrix as the neural network input, Train the model, Load the new data and make prediction.

The model is consisted of few files: functions.R, model_training.R, predict_new_data.R, etc, which will be discussed and illustrated seperately.

Attention! Don't run a heavy command and hit that red button on the top-right corner of the rstudio console! The rstudio might crash and you might have to create a new user.

2. functions.R

This file defines several functions, including importing data from SQL server, tokenizing the sentence, building document term matrix for neural net input. I saved them in a single file in order to call from training and prediction stage with source("function.R").

2.1 load_N_clean

```
load_N_clean <- function(num_of_records) {
  con <- dbConnect(RSQLServer::SQLServer(), server = "TEST", database = 'SparkTest')
  query <- paste("SELECT TOP",as.character(num_of_records),
                 "[BreakDownContent],[QuerySentimentType]
                  FROM [SparkTest].[dbo].[ahwom_content_label]
                 WHERE [A] is not NULL and [B] is not NULL")
  res <- dbSendQuery(con, query)
  df <- dbFetch(res)
  df <- unique(df)
  dbClearResult(res)
  colnames(df) <- c('BreakDownContent', 'SentimentKey')
  df
}
```

What it does:

- Create a connection to SQL Server

- Making Query and Fetch it to a data frame
- Unique lines will be taken and NAs will be removed(There is Actually no NA)
- Regulate column names of the data frame: BreakDownContent, SentimentKey
- Will return a data frame with clean data for vectorization

What can be adjusted:

- num_of_recodes: this is an input argument for the function. It control the loading size of the data.
- The dbConnect: this connection can be substitude with a different SQL server, the server configuration is saved in sql.yaml.
- The SQL query: the second argument of dbSendQuery is the query sent to the database, and change it to obtain different data. Please might the sythax.

2.2 get_dailyDF

```
get_dailyDF <- function() {
  query <- 'SELECT [BreakDownID],[BreakDownContent]
  FROM [SparkTest].[dbo].[AutohomePRCBreakDown_Spark]
  where convert(date,createtime)=convert(date,getdate())'
  con <- dbConnect(RSQLServer::SQLServer(), server = "TEST", database = 'SparkTest')
  res <- dbSendQuery(con,query)
  daily_df <- dbFetch(res)
  dbClearResult(res)
  daily_df
}
```

This function make a query to the same server, getting the daily data for prediction.

2.3 removewords

```
removewords = function(target_words,stop_words){
  target_words <- target_words[target_words%in%stop_words==FALSE]
  return(target_words)
}
```

What is does

- Remove stop words function
- Input: sentence token, stop words dictionary
- Output: sentence token with stop words removed

Just leave it alone.

2.4 sentence_token

```
sentence_token <- function(sentence) {
  # Input all kinds of existing dictionaries
  A_words <- readLines("A_words.txt", encoding = 'UTF-8')
  B_words <- readLines("B_words.txt", encoding = 'UTF-8')
  stopping_words <- readLines("stopping_words.txt", encoding = 'UTF-8')

  # Merge dictionaries
  mydict <- c(A_words,B_words)

  # Pick the unique words
  mydict <- unique(mydict)

  # Remove numbers from the sentence
  sentence <- removeNumbers(sentence)
  # Initialize the segment engine with user dictionary
  engine <- worker()
  new_user_word(engine,mydict)

  # English . will cause jiebaR reporting error
  sentence <- gsub("\\.","",sentence)

  # Sentence segment without stopping words
  segwords <- sapply(sentence, segment, engine)

  # Remove stopping words
  return(sapply(segwords, removewords, stopping_words))
}
```

This is a Chinese sentence tokenizer with jiebaR segment engine embedded. Customized dictionaries are saved locally under this working directory.

What it does:

- Tokenize the sentences, remove the stopping words by customized user dictionary
- Built for DTM creation, no direct usage needed.

What can be adjusted:

- The input sentence must be a List of characters, data frame will not be accepted.

2.5 get_dtm

```
get_dtm <- function(df, count) {
  sentence <- df$BreakDownContent
  # Create the text2vec chinese token
  it <- itoken(sentence,
               tokenizer = sentence_token,
               progressbar = FALSE)
```

```
  # Build a text2vec vocabulary with n gram
  vocab <- create_vocabulary(it, ngram = c(1L, 2L))

  # Adjust the minimum count here, reduce the dimension
  vocab <- vocab %>% prune_vocabulary(term_count_min = count,
                                      doc_proportion_max = 0.5)
  # Build the vectorizer function based on the vocabulary
  vectorizer <- vocab_vectorizer(vocab)

  # Build document-term matrix(DTM), and combine with Sentiment Key
  dtm <- create_dtm(it, vectorizer)

  # Transform labels into number
  df$SentimentKey[which(df$SentimentKey == "NoRec")] <-  0
  df$SentimentKey[which(df$SentimentKey == "Positive")] <- 1
  df$SentimentKey[which(df$SentimentKey == "Negative")] <- -1
  df$SentimentKey <- as.numeric(df$SentimentKey)

  # Combine DTM matrix with labels
  dtm <- cbind(dtm,df$SentimentKey)

  list(dtm,vocab)
}
```

What is does:

- Vectorize the sentence by calling tokenization funtion
- Build document-term matrix(DTM), and combine with Sentiment Key
- The vocabulary is built on 2-gram
- A list of dtm and vocabulary will be returned
- The dtm has labels on the last column(-1,0,1 for Neg,Norec,Pos)
- The vocabulary is used to build dtm with same dimension for new incoming sentences

What can be adjusted:

- Input: Sentence with labels (BreakDownContent, SentimentKey), prune-vocab min count
- ngram = c(1L, 2L) can be changed to ngram = c(1L, nL) to gain n gram vocabulary
- prune-vocab min count can be changed to filter the vocabulary with desired term frequency

2.6 newDT_dtm

```
newDT_dtm <- function(newDT, vocab) {
  vecotorizer <- vocab_vectorizer(vocab)
  newDT <- newDT$BreakDownContent
  it <- itoken(newDT,
               tokenizer = sentence_token,
               progressbar = FALSE)
  dtm <- create_dtm(it, vectorizer)
}
```

What it does:

- Using the same vectorizer from same vocabulary to create a dtm matrix without label(for prediction)

What can be adjusted:

- newDT is the sentences without label, must be a data frame with sentences column name "BreakDown-Content"
- vocab is the vocabulary used to build dtm for the training frame. It's necessary to keep the same dimension

What is a DTM?

- D1 = "I like databases"
- D2 = "I hate databases",

then the document-term matrix would be:

	I	like	hate	databases
D1	1	1	0	1
D2	1	0	1	1

Figure 1:

3.model_training.R

In this part, a h2o.deeplearning model will be implemented based on the pre-built bag of words model.

Data will be read from SQL server, convert into DTM, and the sparse DTM will be write out and read in as H2O frame for the training purpose.

The whole data will be splited by 80% vs 20% for train and test sets. The validation set is omitted for a 4-folds cross-validation will be employed in the deeplearning model.

An introduction of neuron network can be found here for easy understanding: https://github.com/cazala/synaptic/wiki/Neural-Networks-101

The code will be discussed by parts below.

3.1 Data loading and model training

```
# Object: Input data -> Data clean up -> Tokenization -> Vectorization(By BOW)
# -> Train a neural network model
```

```r
# Set working directory and seed
setwd("~/Sentiment_Model")
set.seed(13)

# Load required libraries
library(DBI)
library(RSQLServer)
library(readr)
library(jiebaR)
library(text2vec)
library(h2o)
library(tm)
library(sparsio)

# Call pre-defined funtions
source("functions.R")
```

Leave this part alone, no need to change anything.

```r
# Define the customized parameter
num_of_records <- "50000"
min_word_count <- 5

# Load the training data(For some reason, the df is not constant,
# save the working space everytime you have a new model,
# in order to maintain the vocabulary)
df <- load_N_clean(num_of_records)

# Build the DTM matrix
dtm_N_vocab <- get_dtm(df, min_word_count)
dtm <- dtm_N_vocab[[1]]
vocab <- dtm_N_vocab[[2]]

# write dtm out to read faster in h2o
write_svmlight(x = dtm, file = "dtm.txt", zero_based = FALSE)

# Initialize h2o
h2o.init(nthreads = -1)

# Read the data into h2o
hf <- h2o.importFile("dtm.txt")

# Seperate the training and test group, and making label factor
nrow = dim(hf)[1]
ncol = dim(hf)[2]
hf[,ncol] = h2o.asfactor(hf[,ncol])
sp = h2o.splitFrame(hf,ratios = 0.8)
hf_train = sp[[1]]
hf_test = sp[[2]]
```

In this part num_of_records can be adjusted for smaller or larger sample. Note that the sample might be dupilcated and load_N_clean will remove the duplication. For the exact sample size, please call "dim(df)".

Min_word_count controls the filter for the vocabulary, ie, desired word frequency. For the total input sentence, if a certain word has less frequency than this number, it won't be included in the vocabulary. The

length of the vocabulary will be the number of columns of the neural net input. Call "vocab" to check the dimension.

hf_train and hf_test stand for the train and test set in h2o by the way. The label is on the last column of the matrix.

```
# Now train the model, for detailed documentation, ?h2o.deeplearning
time_NN4x50 = system.time(
  (md_NN4x50 = h2o.deeplearning(1:(ncol - 1), ncol,
                                training_frame = hf_train,
                                nfolds = 4, stopping_metric = "AUTO",
                                model_id = "NN_4x50",
                                stopping_rounds = 3,
                                stopping_tolerance = 1e-3,
                                input_dropout_ratio = 0.2,
                                hidden_dropout_ratios = c(0.3,0.3,0.3,0.3),
                                activation = "RectifierWithDropout",
                                hidden = c(50,50,50,50),
                                reproducible = TRUE, seed = 123))
)
```

Now we train the model. Those arguments will be discussed below:

For more information, please refer "?h2o.deeplearning"

First and second argument(Don't change)

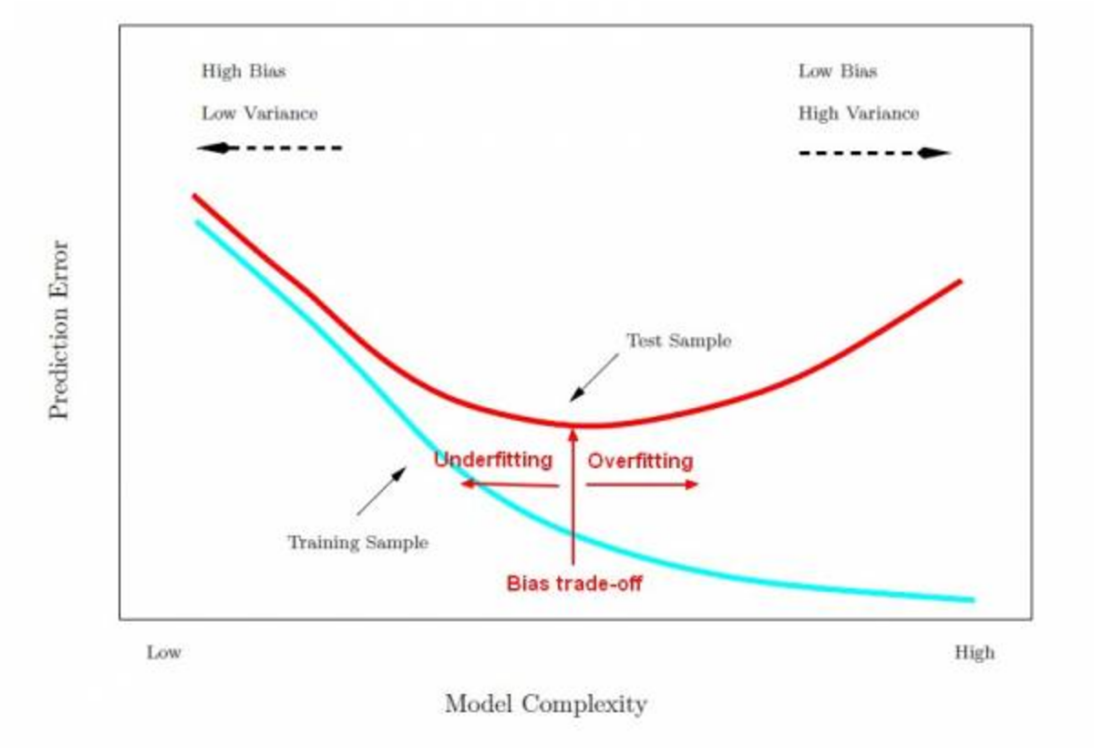
- 1:(ncol - 1), ncol
- Those stand for the columns for predictors and dependent variable, ie, x and y

training_frame = hf_train(Don't change)

- This stands for the input matrix
- Unless the name of input matrix is changed, don't modify this

nfolds = 4

- This stands for the number of folds for cross-validation
- Main purpose of this is preventing over training
- What is over-training? The model will "REMEMBER" the training data and perform badly on other data



stopping_metric, stopping_rounds and stopping_tolerance

- Those stand for stop training if some threshold is met, ie, the model doesn't increase its performance
- This will not improve the model significantly
- Main for time saving
- metric, rounds and tolerance can be justified(but it's just fine for my setting)

input_dropout_ratio, hidden_dropout_ratios

- The key idea is to randomly drop units (along with their connections) from the neural network during training
- This prevents units from co-adapting too much
- It's an approach to prevent the model from overfitting
- Value can be adjusted, the hidden dropout ratios should be a list with the same length of the hidden layer
- $0 <$ ratio < 1

activation

- This is the activation function name
- It calculates a "weighted sum" of its input, adds a bias and then decides whether it should be "fired" or not
- Non-Linearity is added otherwise the stacks of neurons can still be a linear function
- Can be modified. It accepts following funcitons: "Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout", "Maxout", "MaxoutWithDropout". If drop out ratio is provided, the activation must have "WithDropout"
- Details(VPN might be required): https://medium.com/the-theory-of-everything/understanding-activation-functions-in-n

hidden

- This is a list of neuron network structure, and c(50,50,50,50) means that the network has 4 layers, and each layer has 50 neurons.
- Can be modified for different structure. But the suggestion is increasing the neuron to a very large number doesn't leverage the performance dramatically, and keeping it in a simple structure is not a bad choice.(Logistic regression sometime is a good model)

seed and reproducible

- Those are set for reproducebility: giving same input, the same model will be returned.

Other argument

- Please refer to "?h2o.deeplearning" for the list of arguments.

3.2 Model performance and saving work space

It might take a few hours to finish the training, and now we have our model. But what's the performance of the model?

The best way to investigate the performance of a model is input the test data and check the metrics. The test data set is hidden so that the model didn't learn anything from it.

```
h2o.performance(md_NN4x50,hf_test)
```

The metrics and confusion matrix will tell the performace of this model. For simplicity, small MSE(mean square error) is an indicator for a good model, and the error rate from the confusion matrix will give a rough idea of the accuracy.

Do note that accuracy is not a good metrics for a model, as the accuracy is data-dependent. Given different input data, the accuracy might vary.

```
# Save model
h2o.saveModel(md_NN4x50, path = "NN4x50")

# Save the working space, you will need vocabulary to create DTM for the new incoming data
save.image("Second_work_space.RData")
```

The model and work space are saved for future call of prediction.

4. predict_new_data.R

This R script will read new daily short sentence, load the vocabulary according to the model, create dtm, import into h2o and make prediction.

```
# Set working directory and seed
setwd("~/Sentiment_Model")
set.seed(13)

# Call pre-defined funtions
source("functions.R")

# Load required libraries
```

```
library(DBI)
library(RSQLServer)
library(readr)
library(jiebaR)
library(text2vec)
library(h2o)
library(tm)
library(sparsio)

# Initialize h2o
h2o.init(nthreads = -1)
```

Leave this part alone.

```
# Load the previous working space
# Need same vocabulary to create dtm for new data
load("First_work_space.RData")

# Get new data
daily_df <- get_dailyDF()

# Create vectorizer for dtm
vectorizer <- vocab_vectorizer(vocab)

# Create the DTM for the new data
new_dtm <- newDT_dtm(daily_df,vectorizer)

# Write it out for faster sparse matrix reading into h2o
write_svmlight(x = new_dtm, file = "new_dtm.txt", zero_based = FALSE)

# Bring the new dtm as h2o frame
new_hf <- h2o.importFile("new_dtm.txt")

# Load the saved model
my_model <- h2o.loadModel("NN4x50/NN_4x50")
```

The previous work space will be loaded, as we need the vocabulary which is used for build dtm for train frame. Also the model is loaded. If more than 1 model is trained, you might have the variability to choose different work space and model.

Please note that the work space must match the model. Otherwise the num of column of dtm will not match, tons of warning will be returned, and the prediction is useless.

```
# Predictions
pred <- h2o.predict(my_model,new_hf)
pred <- as.data.frame(pred[,1])

# Output the prediction and combine with the sentence
daily_df <- cbind(daily_df,pred)
```

Now the prediction is been made and column bind with the original data.

5. Summary

The whole work consist of 2 parts: vectorization and neural network. And each can be adjusted seperately.

Generally the model works fine for positive and negative sentiment, but it doesn't recognize the neutral emotion. This turns out to be a result of lack of training, as the records of the carefully marked neutral sentiment is insufficient in the train group.

Some suggestion:

- Don't spend a lot of time trying to tune the model. The model won't leverage a huge amount as the benefit of the tuning. And you have no idea to tune which parameter to make it better. It's black magic.
- Carefully marked raw data is important, especially for the neutral sentiment.
- Adding grams might be helpful. 2-grams is employed for now, 3-grams should give the model some consistency, but the minimun word frequency should be adjusted accordingly.