Exploration of the Yelp Dataset with R, SQL and Spark

Crumley

The Data

Our data comes as 7 SQL tables (about 13GB total). We'll use the **RODBC** package in R to communicate with SQL. The first table is a collection of 5261688 user reviews of businesses from Yelp users:

```
yelp_review.df = query.Review("

SELECT TOP 5 * from yelp_review;

")
```

review_id	user_id	business_id	stars	date	text	useful	funny	cool
vkVSC...	bv2nC...	AEx2S...	5	2016-...	Super simple place b...	0	0	0
n6QzI...	bv2nC...	VR6Gp...	5	2016-...	Small unassuming pla...	0	0	0
MV3Cc...	bv2nC...	CKC0-...	5	2016-...	Lester's is located ...	0	0	0
IXvOz...	bv2nC...	ACFtx...	4	2016-...	Love coming here. Ye...	0	0	0
L_9BT...	bv2nC...	s2I_N...	4	2016-...	Had their chocolate ...	0	0	0

The Data

The second is a table of 174567 businesses and their attributes:

```
yelp_business.df = query.Rest("

SELECT TOP 5
    business_id,
    name,
    city,
    stars,
    categories
from yelp_business;

")
```

business_id	name	city	stars	categories
FYWN1...	Denta...	Ahwat...	4	Dentists;General Dentistry;Hea...
He-G7...	Steph...	McMurray	3	Hair Stylists;Hair Salons;Men'...
KQPW8...	Weste...	Phoenix	1.5	Departments of Motor Vehicles;...
8DShN...	Sport...	Tempe	3	Sporting Goods;Shopping
PfOCP...	Brick...	Cuyah...	3.5	American (New);Nightlife;Bars;...

The Goal: Predict the Star Rating of a Review from its Text

Some things to keep in mind:

1. There is a great deal of subjectivity involved in a rating, and even in what a given star rating *means* (what is the ‘objective’ difference between a 3 and 4 star rating?)
2. Even human subjects tend only to agree about 80% of the time as to whether a comment is a positive or negative one.
3. Standard deviation of *stars* is 1.43; any regression model should have a *RMSE* at least that low. A 5 star rating occurs 42.8% of the time; any classification model should be at least that accurate.

So we should be realistic in our hopes for how our models perform.

Data Exploration

Some summaries:

```
review_summaries.df = query.Review("

SELECT
    distinct_reviews = count(distinct(review_id)),
    distinct_users = count(distinct(user_id)),
    distinct_businesses = count(distinct(business_id)),
    avg_star_rating = avg(cast(stars as Float)),
    sd_star_rating = STDEV(cast(stars as Float))
FROM yelp_review

")
```

distinct_reviews	distinct_users	distinct_businesses	avg_star_rating	sd_star_rating
5261668	1326101	174567	3.728	1.434

Data Exploration

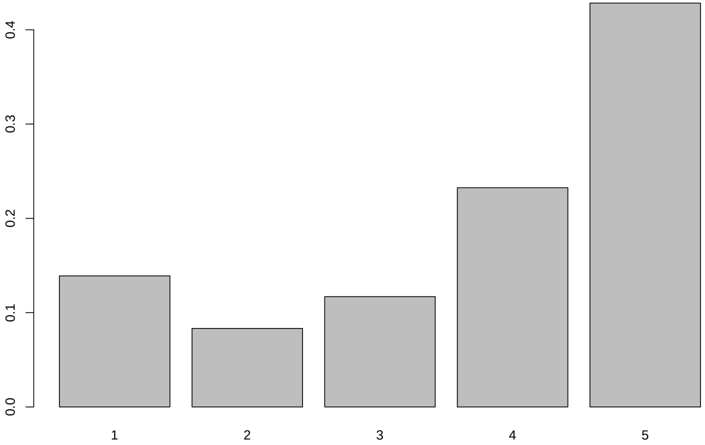
Frequency of star ratings:

```
query.Review("
select
    stars,
    num_reviews = count(*)
into star_frequencies
from yelp_review
group by stars
")

star_frequencies.df = query.Review("
select * from star_frequencies order by stars;
")
```

Data Exploration

```
barplot(star_frequencies.df$num_reviews/sum(star_frequencies.df$num_reviews), names.arg = c(1,2,3,4,5))
```



Our training/test frame will consist of 200,000 randomly chosen reviews, along with the business category for each review.

```
yelp.train = query.Review("
with yelp_joined as (
select
    review_id,
    yelp_review.business_id,
    yelp_review.stars,
    text,
    categories
from yelp_review join yelp_business
    on yelp_review.business_id = yelp_business.business_id
)

select * FROM yelp_joined
WHERE (ABS(CAST(
(BINARY_CHECKSUM(*) *
RAND()) as int)) % 100) < 5
")

yelp.train = yelp.train[1:200000,];
```

review_id	business_id	stars	text	categories
ypjtM...	hjk3o...	1	Food is very bland -...	Vietnamese;Restaurants
rZDvy...	nsWjs...	5	We are lucky to have...	Nightlife;Arts & Ent...
-Km-g...	yuFdJ...	3	Wanted to give this ...	Japanese;Restaurants
SlXVm...	qR62k...	4	very cozy, in the he...	Hotels & Travel;Even...
cd5K0...	pOvTY...	2	I have eaten at seve...	Afghan;Restaurants

First Model: Neural Network

We will extract the adjectives from the reviews using the `openNLP` R package, and then build a document term matrix with the `tm` package (using the most commonly occuring adjectives). These adjectives will serve as the predictors (first layer) of our neural network.

```r
sentence_annotator = Maxent_Sent_Token_Annotator();
word_annotator = Maxent_Word_Token_Annotator();
POS_annotator = Maxent_POS_Tag_Annotator();
extract.POS = function(char.vec, POS.VEC, together = TRUE)  #extract words corresponding
{                                                            #to a given part of speech
  S = as.String(char.vec);
  annotated.char = annotate(S,list(sentence_annotator, word_annotator, POS_annotator));
  type.vec = annotated.char$type;
  word.indices = which(type.vec == "word");
  new.annotated.char = annotated.char[word.indices];
  POS.vec = c();
  for (i in 1:length(new.annotated.char))
  {
    POS.vec = c(POS.vec,new.annotated.char$features[[i]][["POS"]]);
  }
  ans.vec = c();
  for (i in 1:length(POS.VEC))
  {
    pos = POS.VEC[i];
    POS.indices = which(POS.vec == pos);
    start.vec = new.annotated.char$start[POS.indices];
    end.vec = new.annotated.char$end[POS.indices];
    if (length(start.vec) == 0) ans.vec = c(ans.vec,"") else
      ans.vec = c(ans.vec,String(paste.vec(mapply(function (x,y)
        return(substr(S,x,y)), start.vec, end.vec), sep = " ")));
  }
  if (together) return(as.String(paste.vec(ans.vec, sep = " ")))
    else return(unname(lapply(ans.vec, as.String)));
}

extract.POS("The restaurant was quaint, the food was fine, the server was rude.","JJ");
```

```
## quaint fine rude
```

Now that we have just the adjectives, we form a document term matrix, consisting of the top 400 (approximately) most frequently occuring adjectives.

```
library(tm);
adjectives.corpus = Corpus(VectorSource(adjectives.train$adjectives));     #your corpus.
adjectives.DTM = DocumentTermMatrix(adjectives.corpus);    #this is a document term matrix.
dim(adjectives.DTM);      #18551 unique words.  Too many.
adjectives.stemmed <- tm_map(adjectives.corpus, stemDocument);
ncol(adjectives.stemmed.DTM);    #down to 17076 after stemming.  still too many
adjectives.DTM.removed = removeSparseTerms(adjectives.stemmed.DTM,.9975);
ncol(adjectives.DTM.removed);             #now only 442.  Let's use this one.
new.matrix = t(apply(as.matrix(adjectives.DTM.removed), MARGIN = 1,
                   function (x) {n = sum(x); if (n == 0) return(x) else return(x/n);}));
adjectives.train.frame = as.data.frame(cbind(new.matrix, stars = adjectives.train$stars));

#we'll train on 180,000 observations, test on 20,000

adjectives.test.frame = adjectives.train.frame[180001:200000,];
adjectives.train.frame = adjectives.train.frame[1:180000,];
```

free	good	regular	bad	delight	stars
0.0833	0.25	0.0833	0.0000	0.0000	4
0.0000	0.00	0.0000	0.0000	0.0000	4
0.0000	0.00	0.0000	0.2222	0.1111	3
0.0000	0.00	0.0000	0.0000	0.0000	4

Train a Spark-based neural network, hosted on a Microsoft HDInsight Spark cluster. It has 442 input neurons (predictors), 5 output neurons (classifications), and two middle layers of 100 nodes each.

```
options(repos = "https://mran.microsoft.com/snapshot/2018-08-06")
install.packages("sparklyr")

library(sparklyr); library(dplyr); library(RevoScaleR);
source("mikes functions.R");
load("adjectives.train.frame.Rda"); load("adjectives.test.frame.Rda");    #get training/test frames
adjectives_train = adjectives.train.frame; adjectives.train.frame = NULL;
adjectives_test = adjectives.test.frame; adjectives.test.frame = NULL;
adjectives_test_stars = adjectives_test$stars;   adjectives_test$stars = NULL;

cc <- rxSparkConnect(reset = TRUE, interop = "sparklyr");   #start the spark connection
sc <- rxGetSparklyrConnection(cc)                           #start the sparklyr connection

adjectives.train_tbl = copy_to(sc, adjectives_train);   #convert to spark tables
adjectives.test_tbl = copy_to(sc, adjectives_test);

ml_formula = as.formula(paste.vec(c("stars","~",
    paste.vec(names(adjectives_train)[-length(names(adjectives_train))],sep = "+"))));

ml_nn <- ml_multilayer_perceptron(adjectives.train_tbl, ml_formula,    #train the network (about 15 minutes)
    layers = c(442,100,100,5), max_iter = 15, step_size = .03);

test.predictions = sdf_predict(ml_nn, adjectives.test_tbl);     #make predictions on test frame
predict.temp = test.predictions %>% select(predicted_label);
predict.vec = as.data.frame(predict.temp)$predicted_label;     #convert to R data frame

which(predict.vec == adjectives_test_stars);    #compute accuracy
```

First Model: A Neural Network

Results: **BAD**.

Essentially this network "regressed to the mode", making a prediction of "5 stars" for almost every review.

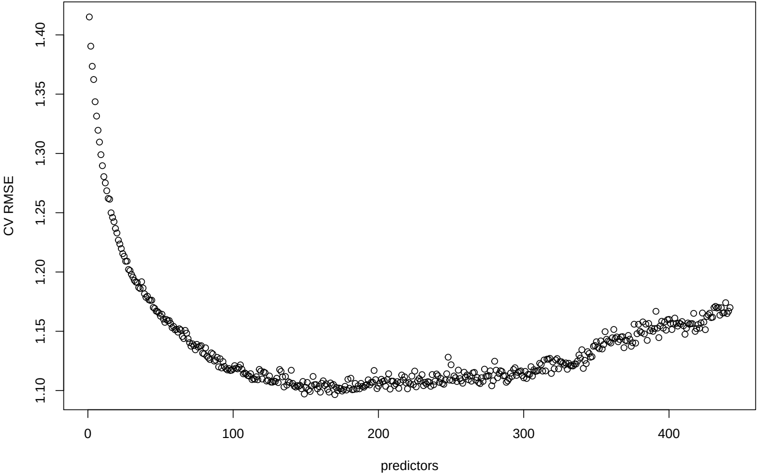
Second Model: Linear Regression with Subset Selection

We will use forward subset selection to identify 442 possible subsets of our adjectives with which to build a linear model. Then we will build each of those models, cross validate each of them, and pick the one with the lowest RMSE.

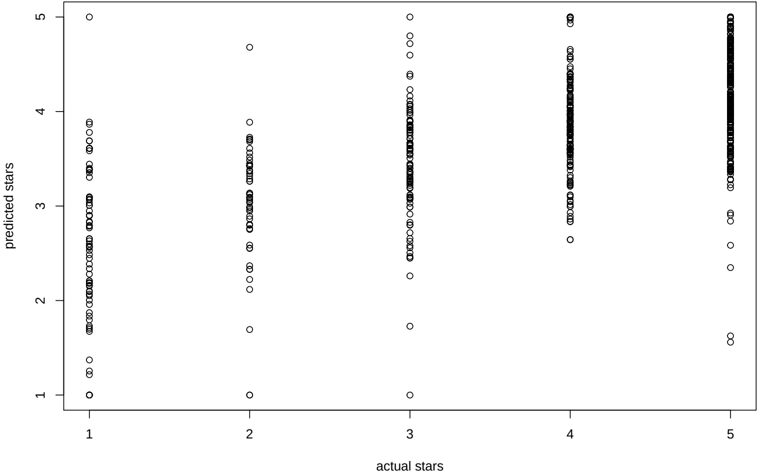
```
library(leaps); library(crossval);

best.predictors = function(nvmax)    #find a candidate subset of predictors of each size
{
  regfit.fwd = regsubsets(stars ~ ., data = adjectives.all, nvmax = nvmax, method = "forward", really.big = TRU
  X = summary(regfit.fwd, matrix.logical = TRUE)$outmat;
  name.vec = colnames(X);
  return(apply(X,MARGIN = 1,function (x) return(name.vec[which(x == TRUE)])));
}
crossval.on.subset = function(predictors)    #cross-validate a subset of predictors
{
  my.formula = paste.vec(c("stars",paste.vec(predictors, sep = " + ")),sep = " ~ ");
  predfun = function(Xtrain, Ytrain, Xtest, Ytest)
  {
    my.model = lm(my.formula, data = as.data.frame(cbind(Xtrain, stars = Ytrain)));
    ypred = predict(my.model, newdata = Xtest);
    return(rmse(ypred,Ytest));
  }
  ans = crossval(predfun, adjectives.all[,-443], adjectives.all[,443], K=10, B = 1, verbose = TRUE)$stat;
  return(ans);
}
```

Optimal number of predictors (adjectives) is around 170, with CV RMSE 1.071, $R^2 = .412$.



Plot of actual stars vs. predicted stars:



Third Model: Sentiment Analysis

It was hoped that our neural network would implicitly learn through training which adjectives were good and bad. This did not happen. So we try a more direct approach.

R has several packages for analyzing the sentiment (positivity or negativity) of natural language sentences. However, they are not all created equal:

```r
library(SentimentAnalysis);
```

```r
analyzeSentiment("That was good.");
```

```
##   WordCount SentimentGI NegativityGI PositivityGI SentimentHE NegativityHE
## 1         1           1            0            1           1            0
##   PositivityHE SentimentLM NegativityLM PositivityLM RatioUncertaintyLM
## 1            1           1            0            1                  0
##   SentimentQDAP NegativityQDAP PositivityQDAP
## 1             1              0              1
```

```r
analyzeSentiment("That was not good.");
```

```
##   WordCount SentimentGI NegativityGI PositivityGI SentimentHE NegativityHE
## 1         1           1            0            1           1            0
##   PositivityHE SentimentLM NegativityLM PositivityLM RatioUncertaintyLM
## 1            1           1            0            1                  0
##   SentimentQDAP NegativityQDAP PositivityQDAP
## 1             1              0              1
```

Third Model: Sentiment Analysis

On the other hand:

```r
library(sentimentr);
```

```r
sentiment("That was good.");
```

```
##    element_id sentence_id word_count sentiment
## 1:          1           1          3 0.4330127
```

```r
sentiment("That was not good.");
```

```
##    element_id sentence_id word_count sentiment
## 1:          1           1          4    -0.375
```

Let's use this one.

Third Model: Sentiment Analysis

We'll feature engineer a new column, the "weighted sentiment" of a review, based on the sentiments of the individual sentences (code not shown).

text	weighted_sentiment	stars
Food is very bland - not authentic at all. meant ...	0.0089881	1
We are lucky to have this venue in Charlotte. Ever...	0.2186977	5
Wanted to give this place a try since it was in my...	-0.2237291	3
very cozy, in the heart of St. Catherine street. ...	0.7825222	4
I have eaten at several of these chains, every tim...	0.1299881	2

Third Model: Sentiment Analysis

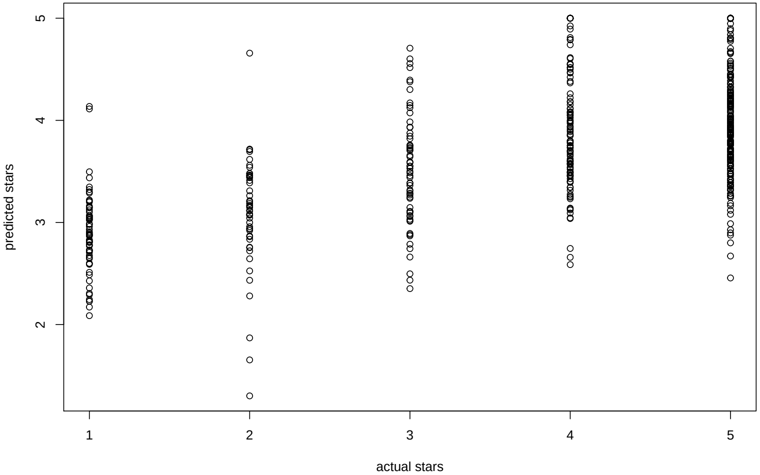
We'll see how a simple linear regression does:

```
##
## Call:
## lm(formula = stars ~ weighted_sentiment, data = review.train[1:180000,
##     ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.7952 -0.8717  0.2124  0.9582  4.6333
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        3.031209   0.003892   778.8   <2e-16 ***
## weighted_sentiment 3.769780   0.014243   264.7   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.218 on 179998 degrees of freedom
## Multiple R-squared:  0.2802, Adjusted R-squared:  0.2802
## F-statistic: 7.006e+04 on 1 and 179998 DF,  p-value: < 2.2e-16
```

Gives 1.21 RMSE on test data (somewhat better than our baseline of 1.43).

Third Model: Sentiment Analysis

Plot of actual vs. predicted star ratings:



Fourth Model: Identify Other Relevant Variables

Could restricting oneself to a certain category of business help to isolate more variability in star rating?

```
model.by.category = function(char.vec)    #perform a linear regression only on certain categories
{
  contains.char = function (x) return(grepl(paste(char.vec, collapse="|"), x));
  my.indices = sapply(review.train$categories, contains.char);
  my.indices = which(my.indices == TRUE);
  df = review.train[my.indices,];
  N = nrow(df);
  df.train = df[1:(floor(.9*N)),];
  df.test = df[-(1:(floor(.9*N))),];
  my.lm = lm(stars ~ weighted_sentiment, data = df.train);
  my.predictions = predict(my.lm, newdata = df.test);
  my.predictions[which(my.predictions > 5)] = 5;
  my.predictions[which(my.predictions < 1)] = 1;
  actual.stars = df.test$stars;
  sample.indices = sample(1:length(my.predictions), min(c(100,length(my.predictions))), replace = FALSE);
  print(summary(my.lm));
  return.val = c(RMSLE = sqrt(1/length(my.predictions)*sum((my.predictions - actual.stars)^2)),
      SDDEV = sd(actual.stars) )
  return(return.val);
}
```

Fourth Model: Identify Other Relevant Variables

```r
model.by.category(c("Restaurant","restaurant"));
```

```
##
## Call:
## lm(formula = stars ~ weighted_sentiment, data = df.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.6473 -0.7988  0.1854  0.9251  4.2033
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)         3.06840    0.00487   630.0   <2e-16 ***
## weighted_sentiment  3.35474    0.01751   191.6   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.172 on 110040 degrees of freedom
## Multiple R-squared:  0.2502, Adjusted R-squared:  0.2502
## F-statistic: 3.672e+04 on 1 and 110040 DF,  p-value: < 2.2e-16
```

```
##    RMSLE    SDDEV
## 1.151035 1.358964
```

Fourth Model: Identify Other Relevant Variables

```r
model.by.category(c("Beauty","beauty"));
```

```
##
## Call:
## lm(formula = stars ~ weighted_sentiment, data = df.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.9652 -0.7967  0.2935  0.9216  4.5107
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)         3.13100    0.01682  186.17   <2e-16 ***
## weighted_sentiment  4.35525    0.05915   73.63   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.229 on 10903 degrees of freedom
## Multiple R-squared:  0.3321, Adjusted R-squared:  0.3321
## F-statistic:  5422 on 1 and 10903 DF,  p-value: < 2.2e-16
```

```
##    RMSLE    SDDEV
## 1.159083 1.451127
```

Fourth Model: Identify Other Relevant Variables

```r
model.by.category(c("Mexican","mexican"));
```

```
##
## Call:
## lm(formula = stars ~ weighted_sentiment, data = df.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.1724 -0.8784  0.1877  0.9540  3.1264
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)         2.99154    0.01611  185.69   <2e-16 ***
## weighted_sentiment  3.61879    0.05810   62.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.205 on 10469 degrees of freedom
## Multiple R-squared:  0.2704, Adjusted R-squared:  0.2703
## F-statistic:  3879 on 1 and 10469 DF,  p-value: < 2.2e-16
```

```
##    RMSLE    SDDEV
## 1.172445 1.400155
```

Fifth Model: What if we restrict ourselves to 1 and 5 star ratings?

We'll select only those ratings that are 1 or 5 star, and perform a logistic regression (binary classification).

```
new.frame = review.train[review.train$stars == 1 | review.train$stars == 5,];
new.frame$stars = factor(new.frame$stars);
train.indices = 1:(floor(.9*nrow(new.frame)));
new.frame.train = new.frame[train.indices,];
new.frame.test = new.frame[-train.indices,];
actual.stars = new.frame.test$stars;

#perform a logistic regression (binary classification)

glm.fit = glm(stars ~ weighted_sentiment, family = binomial, data = new.frame.train);
summary(glm.fit);
test.predictions = predict(glm.fit, type = "response", newdata = new.frame.test);
test.predictions[which(test.predictions > .5)] = "5";
test.predictions[-which(test.predictions > .5)] = "1";
c(test.accuracy = length(which(test.predictions == actual.stars))/length(actual.stars),
               baseline = length(which(actual.stars == 5))/length(actual.stars));
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Call:
## glm(formula = stars ~ weighted_sentiment, family = binomial,
##     data = new.frame.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.7857   0.0016   0.1726   0.4460   4.6256
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -0.50638    0.01271  -39.85   <2e-16 ***
## weighted_sentiment 14.81584    0.10135  146.19   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 114224  on 102326  degrees of freedom
## Residual deviance:  60404  on 102325  degrees of freedom
## AIC: 60408
##
## Number of Fisher Scoring iterations: 6
```

```
## test.accuracy      baseline
##     0.8828496     0.7539138
```

Better.

Some neural networks that actually did perform well

1. Identifying hand drawn digits from their pixels
2. Predicting the cuisine of a recipe from its ingredients