04 - Clustering e Modeling

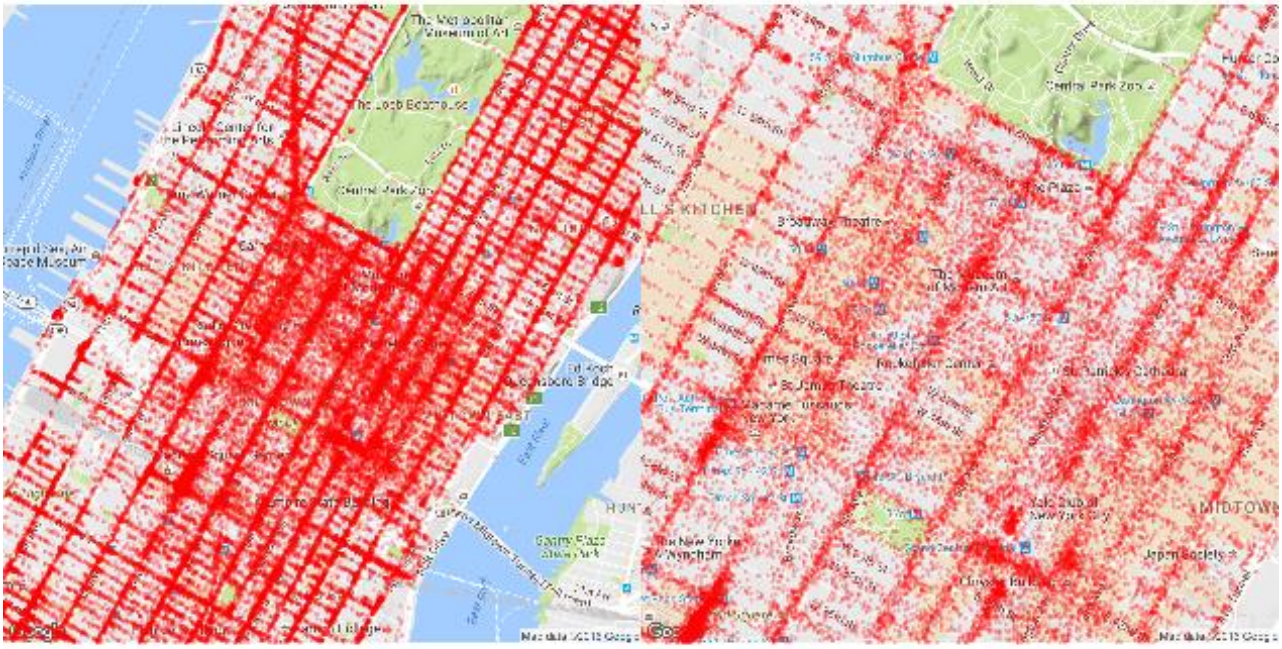
Data Science case study for a Microsoft 213x Project NYC Taxi BigData Analysis
Lorenzo Negri, April 2018
Applications used: Microsoft R Open, Visual Studio, RevoScaleR libraries

Bisogna ora cominciare a preparare i dati per l'analisi raggruppata e la creazione di modelli predittivi.

Clustering

La prima cosa da fare è usare il pacchetto `ggmap` che utilizza l'API di Google per ottenere un paio di mappe. In questo caso, le mappe avranno diversi livelli di zoom.
Quindi una di loro sarà leggermente ingrandita, e l'altra sarà in zoom ad su di un particolare zona di Manhattan.

```
library(ggmap)
map_13 <- get_map(location =  c(lon = -73.98, lat = 40.76), zoom = 13)
map_14 <- get_map(location =  c(lon = -73.98, lat = 40.76), zoom = 14)
map_15 <- get_map(location =  c(lon = -73.98, lat = 40.76), zoom = 15)

q1 <- ggmap(map_14) +
  geom_point(aes(x = dropoff_longitude, y = dropoff_latitude),
             data = mht_sample_df,
             alpha = 0.15,
             na.rm = TRUE, col = "red", size = .5) +
  theme_nothing(legend = TRUE)

q2 <- ggmap(map_15) +
  geom_point(aes(x = dropoff_longitude, y = dropoff_latitude),
             data = mht_sample_df,
             alpha = 0.15,
             na.rm = TRUE, col = "red", size = .5) +
  theme_nothing(legend = TRUE)

require(gridExtra)
grid.arrange(q1, q2, ncol = 2)
```



Dal grafico è possibile individuare sulla mappa dove avvengono con più frequenza le partenze e gli arrivi dei taxi.

Posso ora usare k-means clustering per raggruppare i dati in base alla longitudine e alla latitudine, e da ridimensionare in modo da avere la stessa influenza sui cluster (un modo semplice per ridimensionarli è dividere longitudine per -74 e latitudine per 40). Una volta ottenuti i cluster, possiamo tracciare i centroidi del cluster sulla mappa anziché i singoli punti dati che comprendono ciascun cluster.

```
xydata <- transmute(mht_sample_df,
                    long_std = dropoff_longitude / -74,
                    lat_std = dropoff_latitude / 40)

start_time <- Sys.time()
rxkm_sample <- kmeans(xydata, centers = 300, iter.max = 2000, nstart = 50)
Sys.time() - start_time

# bisogna rimettere i centroidi nella scala originale
centroids_sample <- rxkm_sample$centers %>%
  as.data.frame %>%
  transmute(long = long_std*(-74), lat = lat_std*40, size = rxkm_sample$size)

head(centroids_sample)
```

```
Time difference of 2.017159 mins
```

```
       long      lat size
1 -74.01542 40.71149  277
2 -74.00814 40.71107  443
3 -73.99687 40.72133  335
```

```
4 -74.00465 40.75183  475
5 -73.96324 40.77466  589
6 -73.98444 40.73826  424
```

Nella porzione di codice sopra riportata ho usato la funzione `kmeans` per raggruppare il set di dati campione `mht_sample_df`. In `RevoScaleR` c'è una controparte della funzione `kmeans` chiamata `rxKmeans`, che oltre a lavorare con i `data.frame`, funziona anche con i file XDF. Possiamo quindi utilizzare `rxKmeans` per creare cluster da tutti i dati anziché dal campione rappresentato da `mht_sample_df`.

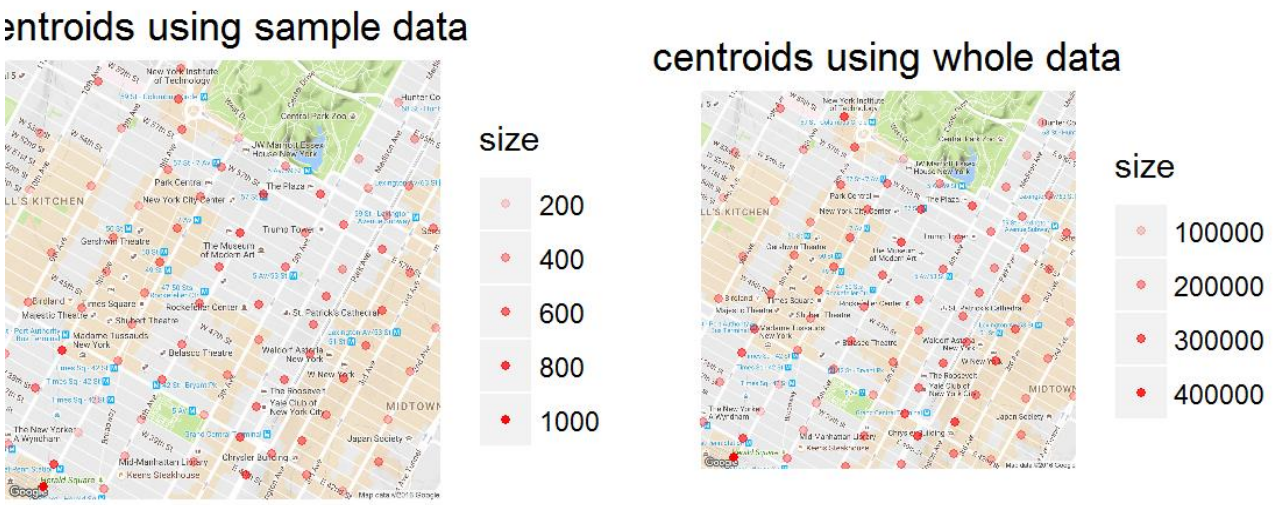
```
start_time <- Sys.time()
rxkm <- rxKmeans( ~ long_std + lat_std, data = mht_xdf,
                  outFile = mht_xdf,
                  outColName = "dropoff_cluster",
                  centers = rxkm_sample$centers,
                  transforms = list(long_std = dropoff_longitude / -74,
                  lat_std = dropoff_latitude / 40),
                  blocksPerRead = 1, overwrite = TRUE,
                  maxIterations = 100, reportProgress = -1)
Sys.time() - start_time

clsdf <- cbind(
  transmute(as.data.frame(rxkm$centers),
            long = long_std*(-74),
            lat = lat_std*40),
  size = rxkm$size, withinss = rxkm$withinss)

head(clsdf)
```

```
Time difference of 2.529844 hours

       long      lat   size      withinss
1 -73.96431 40.80540 301784 0.00059328668
2 -73.99275 40.73042 171080 0.00007597645
3 -73.98032 40.76031 198077 0.00005138354
4 -73.98828 40.77187 134539 0.00011077493
5 -73.96651 40.75752 133927 0.00004789548
6 -73.98446 40.74836 186906 0.00005435595
```

Il mio sistema ha impiegato poco più di due ore e mezza per eseguire tutti i calcoli. Possiamo ora estrarre i centroidi del cluster dall'oggetto risultante e tracciarli su una mappa e confrontare i centroidi calcolati sul sample data con quelli dei dati completi.

```
centroids_whole <- cbind(
  transmute(as.data.frame(rxkm$centers), long = long_std*(-74), lat = lat_std*40),
  size = rxkm$size, withinss = rxkm$withinss)

q1 <- ggmap(map_15) +
  geom_point(data = centroids_sample, aes(x = long, y = lat, alpha = size),
             na.rm = TRUE, size = 1, col = 'red') +
  theme_nothing(legend = TRUE) +
  labs(title = "centroids using sample data")
```

```
q2 <- ggmap(map_15) +
  geom_point(data = centroids_whole, aes(x = long, y = lat, alpha = size),
             na.rm = TRUE, size = 1, col = 'red') +
  theme_nothing(legend = TRUE) +
  labs(title = "centroids using whole data")

require(gridExtra)
grid.arrange(q1, q2, ncol = 2)
```



Come possiamo vedere, i risultati non sono molto diversi, tuttavia esistono differenze e, a seconda del caso d'uso, tali piccole differenze possono avere un grande significato pratico. Se, ad esempio, volessimo scoprire quali sono i luoghi in cui i taxi sono più propensi a lasciare i passeggeri e vietare ai venditori ambulanti di operare in quei punti (per non creare troppo traffico), possiamo fare un lavoro molto più preciso utilizzando i cluster creati utilizzando tutti i dati.

Elaborazione modello per prevedere la variabile tip_percent

Inizo creando un modello lineare che coinvolge due variabili interattive: uno tra *pickup_nb* e *dropoff_nb* e un altro tra *pickup_dow* e *pickup_hour*. L'idea è che vorrei dimostrare che la percentuale di mancia non sia solo influenzata da quale quartiere è stato prelevato il passeggero o a quale quartiere siano stati portati solamente, ma anche da quale quartiere sono stati prelevati e a quale quartiere siano stati portati in correlazione tra loro. Allo stesso modo, vorrei capire se il giorno della settimana e l'ora del giorno possono influenzano la mance nella previsione del modello. Ad esempio, solo perché le persone lasciano molte mance la domenica tra 9 e 12 post meridiane, non è detto che lascino mance alte ogni giorno della settimana tra le 9 e le 12 PM, o in qualsiasi altro momento della giornata di domenica. Questa intuizione è codificata nell'argomento formula del modello che passiamo alla funzione `rxLinMod`: `tip_percent ~ pickup_nb: dropoff_nb + pickup_dow: pickup_hour` dove usiamo `:` per separare i termini interattivi e `+` per separare termini additivi.

```
form_1 <- as.formula(tip_percent ~ pickup_nb:dropoff_nb + pickup_dow:pickup_hour)
rxlm_1 <- rxLinMod(form_1, data = mht_xdf, dropFirst = TRUE, covCoef = TRUE)
```

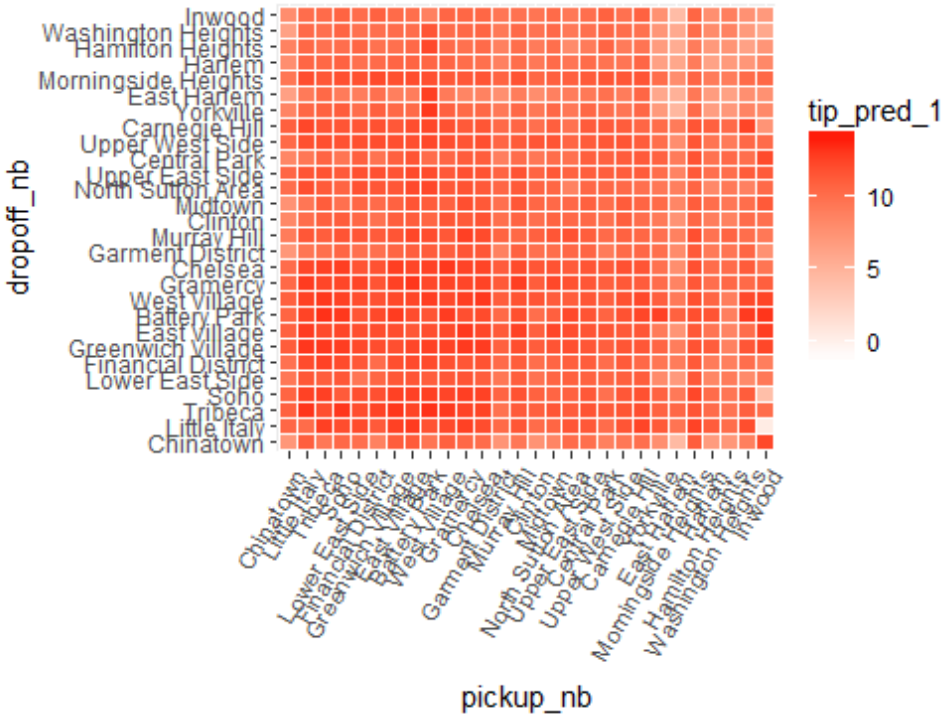
Esaminare i coefficienti del modello individualmente è un compito noioso a causa della quantità. Inoltre, quando si lavora con Big datasets, molti coefficienti vengono rilevati come statisticamente significativi in virtù di una grande dimensione del campione, senza necessariamente essere praticamente significativi. Invece per ora guarderò solo come stanno le nostre previsioni. Inizio estraendo i livelli di ciascuna variabile categorica in una lista che possiamo passare a `expand.grid` per creare un set di dati con tutte le possibili combinazioni dei livelli. Quindi usiamo `rxPredict` per prevedere `tip_percent` usando il modello sopra.

```
rxs <- rxSummary( ~ pickup_nb + dropoff_nb + pickup_hour + pickup_dow, mht_xdf)
ll <- lapply(rxs$categorical, function(x) x[ , 1])
names(ll) <- c('pickup_nb', 'dropoff_nb', 'pickup_hour', 'pickup_dow')
pred_df_1 <- expand.grid(ll)
pred_df_1 <- rxPredict(rxlm_1,
                       data = pred_df_1,
                       computeStdErrors = TRUE,
                       writeModelVars = TRUE)
names(pred_df_1)[1:2] <- paste(c('tip_pred', 'tip_stderr'), 1, sep = "_")
head(pred_df_1, 10)
```

```
   tip_pred_1 tip_stderr_1          pickup_nb dropoff_nb pickup_dow pickup_hour
1    6.796323   0.16432197          Chinatown  Chinatown        Sun     1AM-5AM
2   10.741766   0.15853956       Little Italy  Chinatown        Sun     1AM-5AM
3    9.150114   0.09162002            Tribeca  Chinatown        Sun     1AM-5AM
4   10.174307   0.09819651               Soho  Chinatown        Sun     1AM-5AM
5    9.706202   0.07365164    Lower East Side  Chinatown        Sun     1AM-5AM
6    8.475197   0.06354026 Financial District  Chinatown        Sun     1AM-5AM
7   10.866035   0.07150005  Greenwich Village  Chinatown        Sun     1AM-5AM
8   10.997276   0.06831955       East Village  Chinatown        Sun     1AM-5AM
9    9.313165   0.12507373       Battery Park  Chinatown        Sun     1AM-5AM
10  10.613802   0.11624956       West Village  Chinatown        Sun     1AM-5AM
```

Ora possiamo visualizzare le previsioni del modello tracciando le previsioni medie per tutte le combinazioni delle variabili interattive.

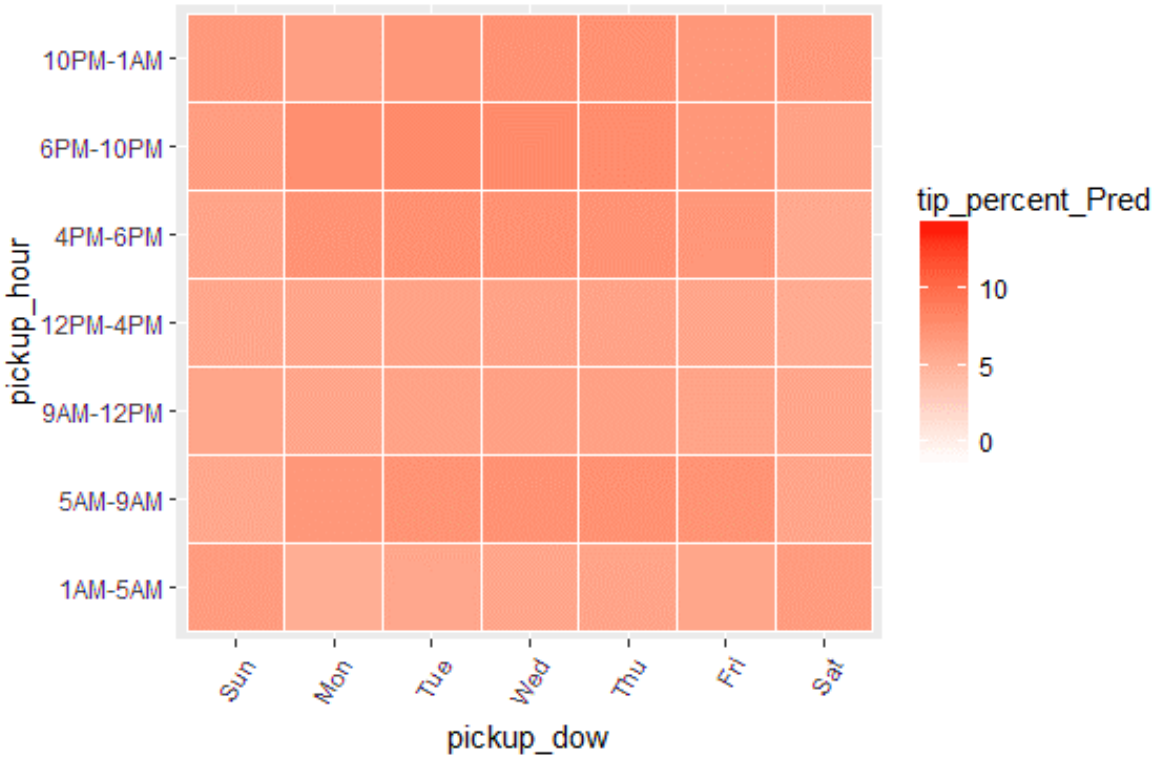
```
ggplot(pred_df_1, aes(x = pickup_nb, y = dropoff_nb)) +
  geom_tile(aes(fill = tip_pred_1), colour = "white") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1)) +
  scale_fill_gradient(low = "white", high = "red") +
  coord_fixed(ratio = .9)
```



Il grafico mostra le percentuali di mancia predette per i quartieri. Fornisce un'idea generale di come il modello distribuisce le mance. Ma non abbiamo una buona visualizzazione di come si comporta.

Il prossimo sarà in combinazione con il giorno della settimana e ora del giorno.

```
ggplot(pred_df_1, aes(x = pickup_dow, y = pickup_hour)) +
  geom_tile(aes(fill = tip_pred_1), colour = "white") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1)) +
  scale_fill_gradient(low = "white", high = "red") +
  coord_fixed(ratio = .9)
```



E ancora una volta possiamo notare alcune tendenze che abbiamo visto anche quando stavamo guardando i dati in precedenza. Vale a dire, i clienti che vengono prelevati durante l'ora di punta del mattino in un giorno feriale o nell'ora di punta del pomeriggio in un giorno della settimana, tendono ad essere generalmente più generosi rispetto alle persone che viaggiano in qualsiasi altro momento della giornata. Quindi alcune di queste tendenze sembrano le stesse ed è un buon segno sapere che il modello è stato capace di catturare parte della variabilità che abbiamo notato nei dati.

Una domanda che potremmo porci adesso sarebbe; quanto è importante l'interazione tra *pickup_dow* e *pickup_hour* nelle previsioni? Quanto peggiorerebbero le previsioni se avessimo mantenuto l'interazione tra *pickup_nb* e *dropoff_nb* e abbandonato la seconda variabile interattiva? Per rispondere a queste domande, possiamo costruire un modello più semplice con `rxLinMod` in cui includiamo solo `pickup_nb:dropoff_nb`.

```
form_2 <- as.formula(tip_percent ~ pickup_nb:dropoff_nb)
rxlm_2 <- rxLinMod(form_2, data = mht_xdf, dropFirst = TRUE, covCoef = TRUE)
pred_df_2 <- rxPredict(rxlm_2,
                       data = pred_df_1,
                       computeStdErrors = TRUE,
                       writeModelVars = TRUE)
names(pred_df_2)[1:2] <- paste(c('tip_pred', 'tip_stderr'), 2, sep = "_")

pred_df <- pred_df_2 %>%
  select(starts_with('tip_')) %>%
  cbind(pred_df_1) %>%
  arrange(pickup_nb, dropoff_nb, pickup_dow, pickup_hour) %>%
  select(pickup_dow,
```

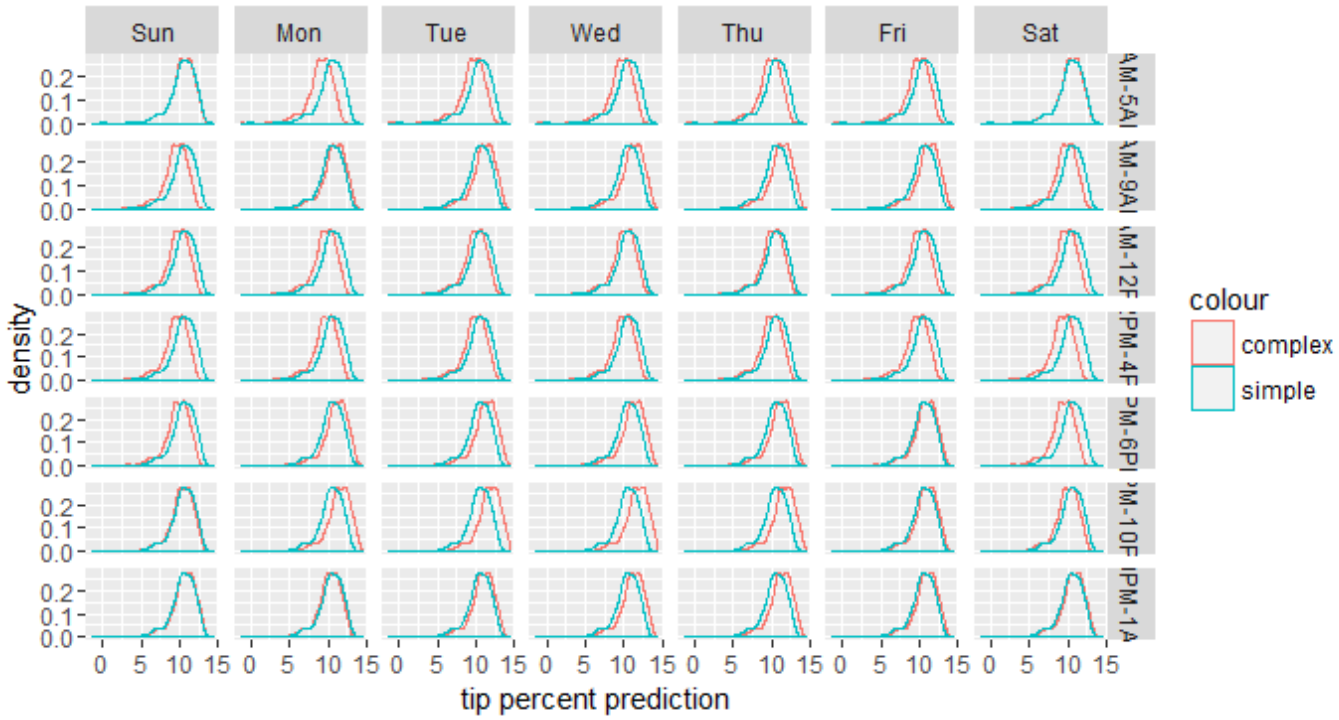
```
          pickup_hour,
          pickup_nb,
          dropoff_nb,
          starts_with('tip_pred_'))

head(pred_df)
```

```
  pickup_dow pickup_hour pickup_nb dropoff_nb tip_pred_2 tip_pred_1
1        Sun      1AM-5AM Chinatown  Chinatown   6.782043   6.796323
2        Sun      5AM-9AM Chinatown  Chinatown   6.782043   5.880284
3        Sun     9AM-12PM Chinatown  Chinatown   6.782043   6.103625
4        Sun     12PM-4PM Chinatown  Chinatown   6.782043   5.913130
5        Sun      4PM-6PM Chinatown  Chinatown   6.782043   6.121957
6        Sun     6PM-10PM Chinatown  Chinatown   6.782043   6.642192
```

Possiamo vedere dai risultati sopra che le previsioni con il modello più semplice sono identiche per tutti i giorni della settimana e tutte le ore per la stessa combinazione pick-up e drop-off. Mentre le previsioni del modello più complesso sono uniche per ogni combinazione di tutte e quattro le variabili. In altre parole, `aggiungendo pickup_dow:pickup_hour` al modello, si aggiunge una variazione extra alle previsioni e ciò che vorremmo sapere è se questa variazione contiene segnali importanti o se si comporta più o meno come semplice rumore.

Per ottenere la risposta, confrontiamo la distribuzione delle due previsioni suddividendole per *pickup_dow* e *pickup_hour*.

```
ggplot(data = pred_df) +
  geom_density(aes(x = tip_pred_1, col = "complex")) +
  geom_density(aes(x = tip_pred_2, col = "simple")) +
  facet_grid(pickup_hour ~ pickup_dow)
```



Il modello più semplice mostra la stessa distribuzione per tutto il grafico, perché queste due variabili non hanno alcun effetto sulle sue previsioni, ma il modello più complesso mostra una distribuzione leggermente diversa per ciascuna combinazione di *pickup_dow* e *pickup_hour*, sotto forma di un leggero scostamento nella distribuzione (simile ad un effetto stereoscopico). Quel effetto è dato da *pickup_dow* e *pickup_hour* ad ogni combinazione delle due variabili. Poiché lo scostamento direzionale (non casuale), cattura un qualche tipo di segnale importante (anche se il suo significato pratico è ancora poco chiaro). Possiamo semplificare il grafico se applichiamo qualche logica di business ad esso.

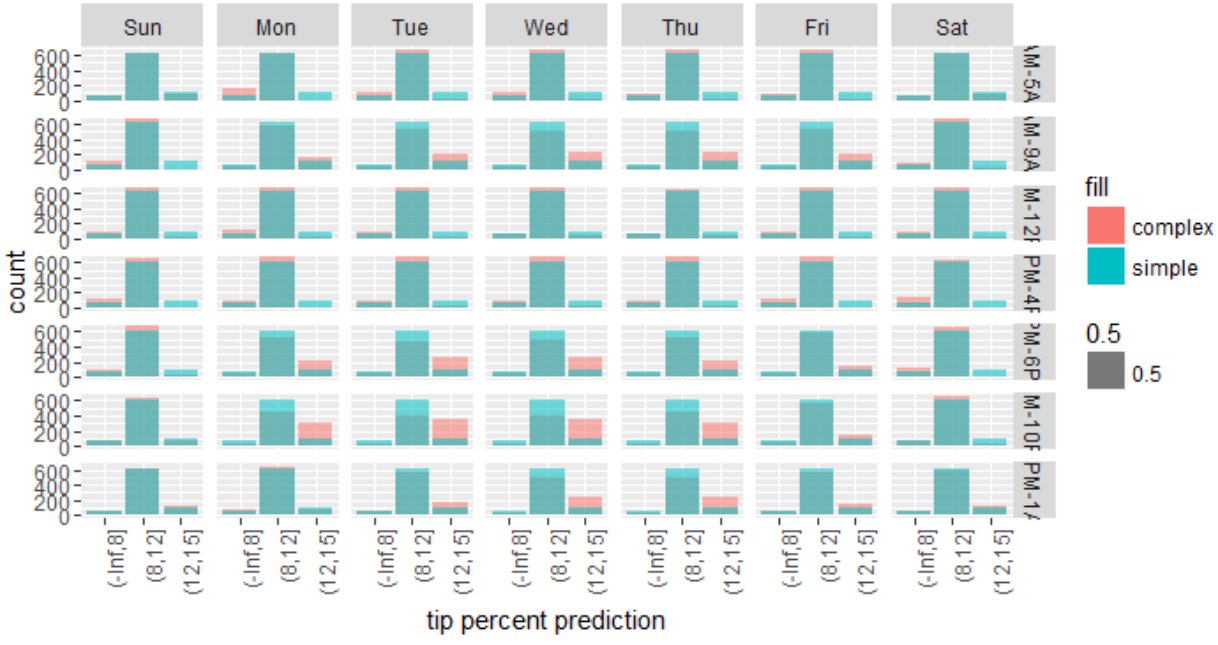
Utilizzo `cut` per raggruppare le previsioni. Per scegliere i *cut-off*, posso usare la funzione `rxQuantile e vedere le distribuzioni`.

```
rxQuantile("tip_percent", data = mht_xdf, probs = seq(0, 1, by = .05))
```

```
 0%   5%  10%  15%  20%  25%  30%  35%  40%  45%  50%  55%  60%  65%  70%  75%  80%
 -1    0    0    0    0    0    0    0    9   12   15   17   17   17   18   18   19
85%  90%  95% 100%
 20   21   23   99
```

In base ai risultati sopra riportati, possiamo raggruppare *tip_percent* quando; è inferiore all'8%, tra l'8% e il 12%, tra il 12% e il 15%, tra il 15% e il 18% o il 18% o più. Possiamo quindi tracciare un grafico a barre che mostri le stesse informazioni di prima, ma leggermente più facile da interpretare.

```
pred_df %>%
  mutate_at(vars(tip_pred_1, tip_pred_2), funs(cut(.,
                                                   c(-Inf, 8, 12, 15, 18, Inf)))) %>%
  ggplot() +
  geom_bar(aes(x = tip_pred_1, fill = "complex", alpha = .5)) +
  geom_bar(aes(x = tip_pred_2, fill = "simple", alpha = .5)) +
  facet_grid(pickup_hour ~ pickup_dow) +
  xlab('tip percent prediction') +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Sulla base del grafico sopra, possiamo vedere che rispetto al modello semplice, il modello complesso tende a prevedere più clienti taxi generosi di mance e meno quelli di medio livello durante determinate combinazioni di giorno e di tempo (come dal lunedì al giovedì durante le ore di punta).

Finora abbiamo analizzato i modelli senza preoccuparci della loro efficacia di previsione. Per verificarlo abbiamo bisogno di suddividere il dataset in due parti una per il test delle previsione e un altro per l'apprendimento.

Per dividere i dati, prima userò `rxDataStep` per creare una nuova colonna chiamata *split* Per dividere i dati in parti di *training* e *testing*, la nuova colonna avrà ogni riga una dicitura come "*train*" o "*test*" in modo tale che una determinata proporzione dei dati (qui sceglierò il 75 percento) venga utilizzata per l'apprendimento del modello e il resto utilizzato per testare la potenza predittiva. Utilizzo quindi la funzione `rxSplit` per dividere i dati. La funzione `rx_split_xdf` che verrà creata combina i due passaggi in uno e imposta alcuni argomenti sui valori predefiniti.

```
dir.create('output', showWarnings = FALSE)
rx_split_xdf <- function(xdf = mht_xdf,
                         split_perc = 0.75,
                         output_path = "output/split",
                         ...) {

  # prima creo la colonna per la suddivisione
  outFile <- tempfile(fileext = 'xdf')
  rxDataStep(inData = xdf,
             outFile = xdf,
             transforms = list(
               split = factor(ifelse(rbinom(.rxNumRows,
                                            size = 1,

                                            prob = splitperc),
                                     "train", "test"))),
             transformObjects = list(splitperc = split_perc),
             overwrite = TRUE, ...)

  # quindi suddivido i dati in due in base alla colonna che abbiamo appena creato
  splitDS <- rxSplit(inData = xdf,
                     outFilesBase = file.path(output_path, "train"),
                     splitByFactor = "split",
                     overwrite = TRUE)

  return(splitDS)
}

# we can now split to data in two
mht_split <- rx_split_xdf(xdf = mht_xdf, varsToKeep = c('payment_type',
                                                        'fare_amount',
                                                        'tip_amount',
                                                        'tip_percent',
                                                        'pickup_hour',
                                                        'pickup_dow',
```

```
                                                'pickup_nb',
                                                'dropoff_nb'))
names(mht_split) <- c("train", "test")
```

Ora eseguiamo tre diversi algoritmi:

- `rxLinMod`, il modello lineare precedente con i termini `tip_percent ~ pickup_nb:dropoff_nb + pickup_dow:pickup_hour`
- `rxDTree`, l'algoritmo ad albero decisionale con i termini `tip_percent ~ pickup_nb + dropoff_nb + pickup_dow + pickup_hour` (gli alberi decisionali non hanno bisogno di fattori interattivi perché le interazioni sono incorporate nell'algoritmo stesso)
- `rxDForest`, l'algoritmo di foresta casuale con gli stessi termini sopra.

```
system.time(linmod <- rxLinMod(tip_percent ~ pickup_nb:dropoff_nb
                               + pickup_dow:pickup_hour,
                               data = mht_split$train,
                               reportProgress = 0))
system.time(dtree <- rxDTree(tip_percent ~ pickup_nb
                             + dropoff_nb + pickup_dow + pickup_hour,
                             data = mht_split$train,
                             pruneCp = "auto", reportProgress = 0))
system.time(dforest <- rxDForest(tip_percent ~ pickup_nb
                                 + dropoff_nb + pickup_dow + pickup_hour,
                                  mht_split$train, nTree = 10, importance = TRUE,
                                  useSparseCube = TRUE, reportProgress = 0))
```

```
   user  system elapsed
   0.00    0.00    1.62

   user  system elapsed
   0.03    0.00  778.00

   user  system elapsed
   0.02    0.00  644.17
```

Poiché l'esecuzione degli algoritmi possono richiedere tanto tempo, può valere la pena di salvare i modelli così nel caso perdessimo la nostra sessione R o in cui dobbiamo riavviare, non è necessario eseguire i modelli una seconda volta. Possiamo semplicemente recuperarli dalla nostra directory e usarli per fare previsioni. Questo è anche molto utile se creiamo i modelli localmente sui nostri laptop. E poi vogliamo usarli in Hadoop o su di un server SQL o un server più capiente con più dati in esso.

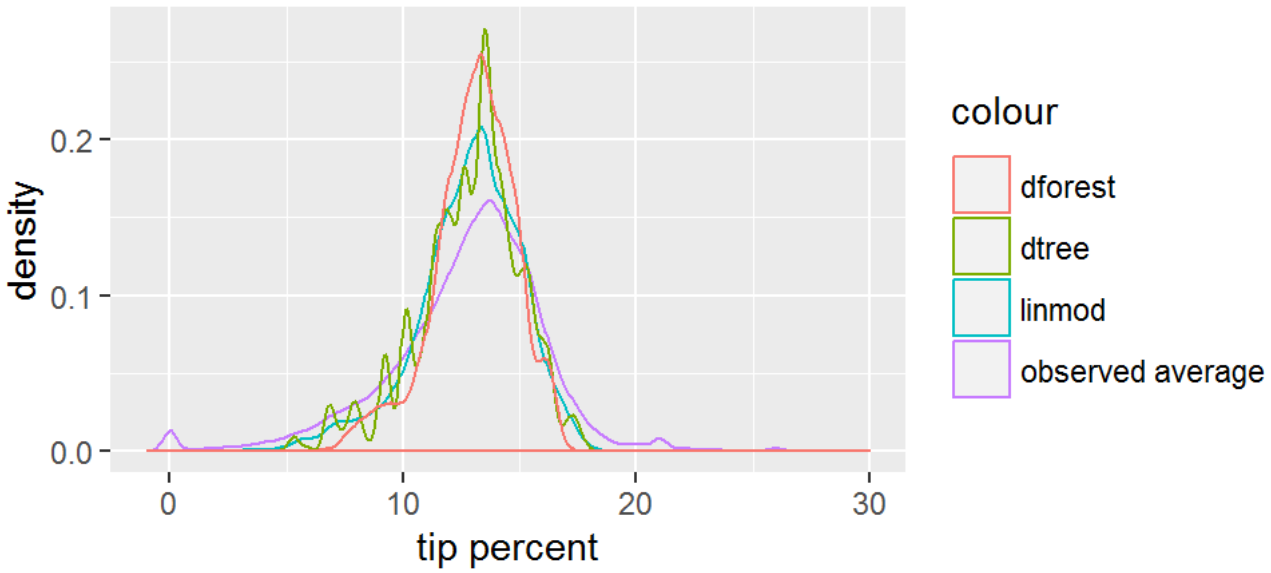
```
trained.models <- list(linmod = linmod, dtree = dtree, dforest = dforest)
save(trained.models, file = 'trained_models.Rdata')
```

Prima di applicare l'algoritmo a tutto il dataset, applico il modello al sample dataset con tutte le combinazioni di variabili categoriali e visualizzo i risultati. Questo potrebbe aiutarci a sviluppare nuove intuizioni migliorative su ciascun algoritmo.

```
pred_df <- expand.grid(ll)
pred_df_1 <- rxPredict(trained.models$linmod,
                       data = pred_df, predVarNames = "pred_linmod")
pred_df_2 <- rxPredict(trained.models$dtree,
                       data = pred_df, predVarNames = "pred_dtree")
pred_df_3 <- rxPredict(trained.models$dforest,
                       data = pred_df, predVarNames = "pred_dforest")
pred_df <- do.call(cbind, list(pred_df, pred_df_1, pred_df_2, pred_df_3))
head(pred_df)
```

```
           pickup_nb dropoff_nb pickup_hour pickup_dow pred_linmod pred_dtree pred_dforest
1          Chinatown  Chinatown     1AM-5AM        Sun 6.869645    5.772054   9.008643
2       Little Italy  Chinatown     1AM-5AM        Sun 10.627190   9.221250   10.634590
3            Tribeca  Chinatown     1AM-5AM        Sun 9.063741    9.221250   10.099731
4               Soho  Chinatown     1AM-5AM        Sun 10.107815   8.313437   10.162946
5    Lower East Side  Chinatown     1AM-5AM        Sun 9.728399    9.221250   10.525242
6 Financial District  Chinatown     1AM-5AM        Sun 8.248997    6.937500   8.674807
```

```
observed_df<-rxSummary(tip_percent~pickup_nb:dropoff_nb:pickup_dow:pickup_hour,
                       mht_xdf)
observed_df <- observed_df$categorical[[1]][ , c(2:6)]
pred_df <- inner_join(pred_df, observed_df, by = names(pred_df)[1:4])

ggplot(data = pred_df) +
  geom_density(aes(x = Means, col = "observed average")) +
  geom_density(aes(x = pred_linmod, col = "linmod")) +
  geom_density(aes(x = pred_dtree, col = "dtree")) +
  geom_density(aes(x = pred_dforest, col = "dforest")) +
  xlim(-1, 30) +
  xlab("tip percent")
```



Sia il modello lineare che la foresta casuale ci forniscono previsioni fluide. Possiamo vedere che le previsioni *dforest* sono le più concentrate. Le previsioni per *dtree* seguono una distribuzione

frastagliata, probabilmente a causa del sovradattamento (overfitting), ma non lo sappiamo fino a quando non controlleremo le prestazioni rispetto al set di test. Nel complesso, le previsioni sono più ristrette rispetto alla media reale.

Ora applichiamo il modello ai dati del test in modo da poter confrontare il potere predittivo di ciascun modello. Se abbiamo ragione sul fatto che l'algoritmo *decision tree* si adatta in modo eccessivo, dovremmo vederlo preformare peggio nei dati del test rispetto agli altri due modelli. Se crediamo che la foresta casuale catturi alcuni segnali intrinseci nei dati che mancano al modello lineare, dovremmo vederlo funzionare meglio del modello lineare sui dati del test.

La prima metrica che osserviamo è la media dei residui al quadrato, che ci dà un'idea di quanto siano vicine le previsioni ai valori osservati. Dato che prevediamo la percentuale di mancia (*tip percent*), che di solito scende in un intervallo ristretto compreso tra lo 0 e il 20 percento, dovremmo aspettarci in media che i residui per un buon modello non siano più di 2 o 3 punti percentuale.

```
rxPredict(trained.models$linmod,
          data = mht_split$test,
          outData = mht_split$test,
          predVarNames = "tip_percent_pred_linmod",
          overwrite = TRUE)
rxPredict(trained.models$dtree,
          data = mht_split$test,
          outData = mht_split$test,
          predVarNames = "tip_percent_pred_dtree",
          overwrite = TRUE)
rxPredict(trained.models$dforest,
          data = mht_split$test,
          outData = mht_split$test,
          predVarNames = "tip_percent_pred_dforest",
          overwrite = TRUE)

rxSummary(~ SE_linmod + SE_dtree + SE_dforest, data = mht_split$test,
          transforms = list(SE_linmod = (tip_percent - tip_percent_pred_linmod)^2,
                            SE_dtree = (tip_percent - tip_percent_pred_dtree)^2,
                            SE_dforest = (tip_percent
                                          - tip_percent_pred_dforest)^2))
```

```
Call:
rxSummary(formula = ~ SE_linmod + SE_dtree + SE_dforest, data = mht_split$test,
    transforms = list(SE_linmod = (tip_percent - tip_percent_pred_linmod)^2,
        SE_dtree = (tip_percent - tip_percent_pred_dtree)^2,
        SE_dforest = (tip_percent - tip_percent_pred_dforest)^2))

Summary Statistics Results for: ~SSE_linmod + SSE_dtree + SSE_dforest
Data: mht_split$test (RxXdfData Data Source)
File name: C:\Data\NYC_taxi\output\split\train.split.train.xdf
Number of valid observations: 43118543

 Name       Mean     StdDev   Min                 Max     ValidObs MissingObs
 SE_linmod  82.66458 108.9904 0.00000000005739206 9034.665 43118542 1
 SE_dtree   82.40040 109.1038 0.00000251589457986 8940.693 43118542 1
 SE_dforest 82.47107 108.0416 0.00000000001590368 8606.201 43118542 1
```

Un'altra metrica che vale la pena di osservare è una matrice di correlazione. Questo può aiutarci a determinare in quale misura le previsioni dei diversi modelli siano vicine l'una all'altra e in che misura ciascuna si avvicina alla percentuale di mancia effettiva o osservata.

```
rxc <- rxCor( ~ tip_percent
              + tip_percent_pred_linmod
              + tip_percent_pred_dtree
              + tip_percent_pred_dforest, data = mht_split$test)
print(rxc)
```

```
                         tip_percent pred_linmod pred_dtree pred_dforest
tip_percent                1.0000000   0.1391751   0.1500126  0.1499031
tip_percent_pred_linmod    0.1391751   1.0000000   0.8580617  0.9084119
tip_percent_pred_dtree     0.1500126   0.8580617   1.0000000  0.9404640
tip_percent_pred_dforest   0.1499031   0.9084119   0.9404640  1.0000000
```

Possiamo vedere che c'è solo una leggera differenza di correlazione tra i modelli e *tip-percent*. La predizione dal decision tree risulta la migliore. Possiamo anche vedere che la correlazione tra le previsioni sono abbastanza vicine tra loro. La foresta casuale fa previsioni che sono simili al modello lineare e piuttosto vicine a quelle dell'albero decisionale. Quindi sembra che ci troviamo di fronte a una situazione dove abbiamo tre diversi modelli, tutti e tre non stanno facendo un buon lavoro di previsione. Ma tutti e tre comunque stanno facendo predizioni che sono ragionevoli e vicine l'una all'altra. Tutti e tre i modelli non sono ancora elaborati in maniera completa. Possiamo renderli più ricchi aggiungendo altre variabili, aggiungendo altri input.

Dopo alcuni esperimenti i risultati ottenuti con il modello lineare sono:

Linear Model Formula Predictors	*R-sqd (adj)*
tip_percent~trip_duration + pickup_dow : pickup_hour	**< 0.01**
tip_percent~trip_duration + payment_type_desc + pickup_dow : pickup_hour	**> 0.70**

Potete vedere che l'R2 corretto è aumentato parecchio rispetto al modello precedente. Come abbiamo esaminato in precedenza, quasi tutti i pagamenti in contanti mostrano una percentuale zero di mancia, rendendo *payment_type_desc* una caratteristica fondamentale per la previsione.