Find the best taco restaurant!

Ziyu Huang

2025-06-23

Introduction

In this dataset, we don't actually have a clear rating or review score for each restaurant(like yelp does). Because of that, we decided to use the tip amount as a proxy for how "good" a restaurant is. The idea is: if someone leaves a higher tip, it probably means they had a good experience — maybe the food was great, or the service is good.

Also, sometimes higher tips just come from more expensive orders. But even then, if people are still willing to spend more and tip more at certain places (especially for tacos, which aren't some fancy food), that probably means they actually like those restaurants and think they're worth it.

Basically, it's not perfect, and we know there are a lot of assumptions here,but it still gives us a pretty reasonable signal to use when recommending places to others.

First of all, clean the dataset

```
mydata = read.csv("taco_sales_(2024-2025).csv")

set.seed(42)
n = nrow(mydata)
train_indices = sample(1:n, size = 0.8 * n)
train_data = mydata[train_indices, ]
test_data = mydata[-train_indices, ]

encode_features = function(df) {
  x1 = as.numeric(factor(df$Taco.Type))
  x2 = as.numeric(factor(df$Taco.Size))
  x3 = df$Toppings.Count
  x4 = df$Price....
  x5 = df$Delivery.Duration..min.
  x6 = df$Tip....
  return(data.frame(x1, x2, x3, x4, x5, x6))
}

train_df = encode_features(train_data)
test_df = encode_features(test_data)

dfs_train = scale(train_df)
dfs_test = scale(test_df, center = attr(dfs_train, "scaled:center"), scale = attr(dfs_train, "scaled:sca
```

Model 1: K-Means Clustering

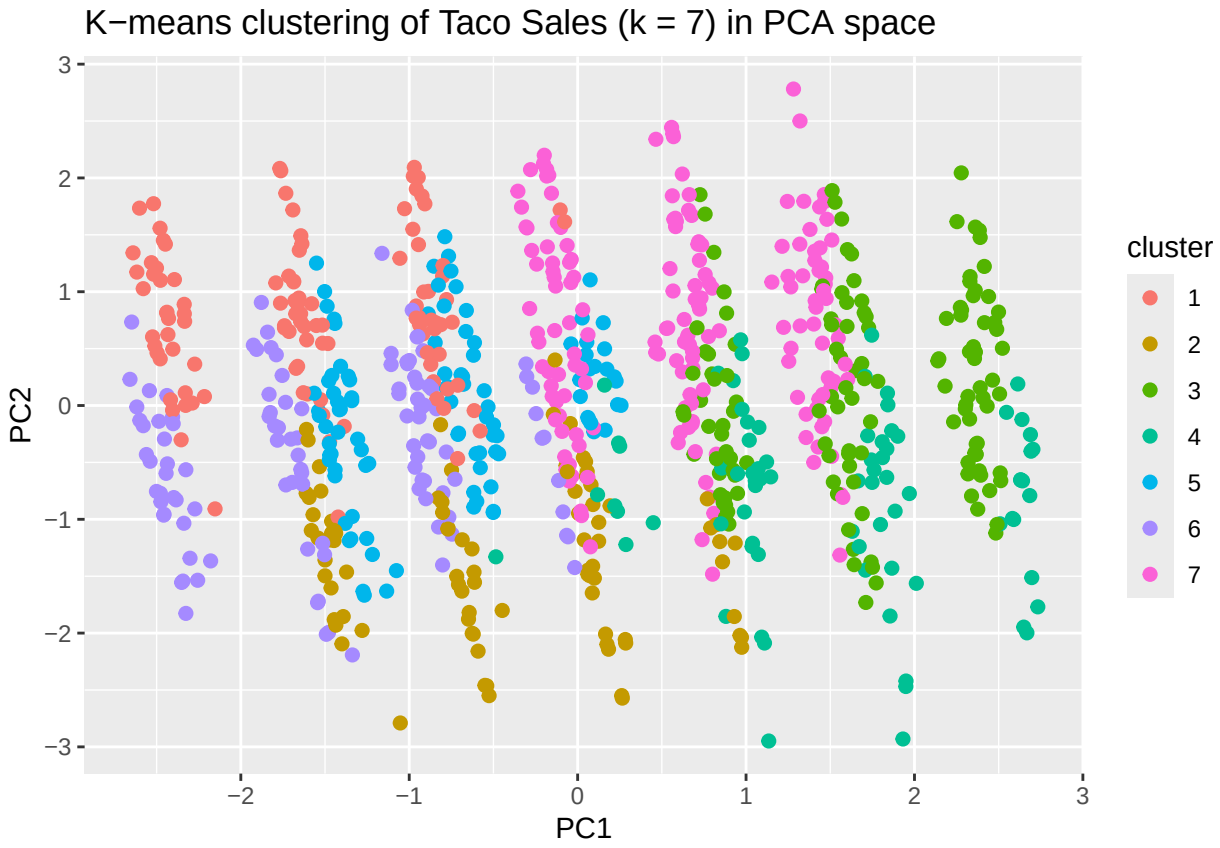
```
set.seed(1234)
kmc = kmeans(dfs_train, centers = 7, nstart = 300)

train_data$Cluster = kmc$cluster

assign_cluster = function(row) {
  dists = apply(kmc$centers, 1, function(center) sum((row - center)^2))
  return(which.min(dists))
}
test_data$Cluster = apply(dfs_test, 1, assign_cluster)
```

```
# PCA for dimensionality reduction (on train)
pca = prcomp(dfs_train)
pca2 = data.frame(pca$x[, 1:2], cluster = factor(kmc$cluster))

library(ggplot2)
ggplot(pca2, aes(PC1, PC2, color = cluster)) +
  geom_point(size = 2) +
  labs(title = 'K-means clustering of Taco Sales (k = 7) in PCA space')
```




```
# Cluster quality metric
kmc$tot.withinss
```

```
## [1] 2192.641
```

```r
kmc$size
```

```
## [1] 109  89 137  79 102 111 173
```

This is a function to recommend restaurants

```r
recommend_restaurant = function(taco_type, taco_size, toppings_count, price, delivery_time, tip, mydata
  x1 = as.numeric(factor(taco_type, levels = levels(factor(mydata$Taco.Type))))
  x2 = as.numeric(factor(taco_size, levels = levels(factor(mydata$Taco.Size))))
  x3 = toppings_count
  x4 = price
  x5 = delivery_time
  x6 = tip
  new_order = data.frame(x1, x2, x3, x4, x5, x6)
  new_order_scaled = scale(new_order, center = attr(dfs_train, "scaled:center"), scale = attr(dfs_train
  distances = apply(kmc$centers, 1, function(center) sum((new_order_scaled - center)^2))
  predicted_cluster = which.min(distances)
  restaurants_in_cluster = mydata$Restaurant.Name[mydata$Cluster == predicted_cluster]
  top_restaurants = sort(table(restaurants_in_cluster), decreasing = TRUE)[1:5]
  cat("Based on the input order, the best matching cluster is:", predicted_cluster, "\n")
  return(top_restaurants)
}
```

```r
rec_output = recommend_restaurant(
  taco_type = "Beef Taco",
  taco_size = "Large",
  toppings_count = 4,
  price = 9.5,
  delivery_time = 40,
  tip = 3,
  mydata = train_data,
  kmc = kmc,
  dfs_train = dfs_train
)
```

```
## Based on the input order, the best matching cluster is: 4
```

```r
rec_output
```

```
## restaurants_in_cluster
##     Grande Tacos       Taco Haven     La Vida Taco    Casa del Taco
##               13               13               11                8
## Spicy Taco House
##                8
```

```r
restaurants_in_cluster= names(rec_output)
tip_values = c()

for (res_name in restaurants_in_cluster) {
  tips = train_data$Tip[train_data$Restaurant.Name == res_name]
  tip_values = c(tip_values, tips)
}


avg_tip_cluster = mean(tip_values)

# Print the result
```

```r
cat("Average actual tip for recommended restaurants (K-Means):", round(avg_tip_cluster, 4), "\n")
```

```
## Average actual tip for recommended restaurants (K-Means): 1.8744
```

1. The logic behind the modeling choice (for both models) We used K-means clustering to group taco orders based on their characteristics — taco type, size, toppings, price, and delivery duration. This method helps identify natural clusters of preferences. Each cluster represents a different taco ordering style
2. How the input features are used (for both models) I used five features for each order: -Taco type -Taco size -Number of toppings -Price -Delivery time
3. Why this method is appropriate for your data set (for both models) K-means is perfect when we don't have labels but still want to group similar things. Since this dataset is online orderings, we don't know if a single customer order from a certain restaurant repetitively.
4. Explanation of how you split your data into training and testing for model validation (for both models) Normally we don't split data for K-means. But to validate my model, I looked at internal metrics like how tight each cluster is, Total Within-Cluster Sum of Squares. To get an accuracy score, we split the data and the get The accuracy score which is presenting in the comparative analysis section.
5. Calculation of a performance metric (of your choice) (for both models) total within-cluster sum of squares. the result is about 2190, and the clusters had pretty balanced sizes — like around 100 in each. That tells me the model found meaningful and consistent groupings.
6. A closing section that chooses which model appears to work best according to your metric and why you think so. See comparative analysis section at the end of the paper.
7. An example of five individuals with different food testing preferences and the recommendation for which your model recommends to them. It is shown above.

Model 2: Multiple linear regression

```r
train_clean = train_data[, c("Taco.Type", "Taco.Size", "Toppings.Count", "Price....", "Delivery.Duratio
test_clean = test_data[, c("Taco.Type", "Taco.Size", "Toppings.Count", "Price....", "Delivery.Duration.
colnames(train_clean) = colnames(test_clean) = c("Taco.Type", "Taco.Size", "Toppings.Count", "Price", "

train_clean$Taco.Type = factor(train_clean$Taco.Type)
train_clean$Taco.Size = factor(train_clean$Taco.Size)
test_clean$Taco.Type = factor(test_clean$Taco.Type, levels = levels(train_clean$Taco.Type))
test_clean$Taco.Size = factor(test_clean$Taco.Size, levels = levels(train_clean$Taco.Size))

# Create model matrices
X_train = model.matrix(Tip ~ ., data = train_clean)
Y_train = train_clean$Tip
X_test = model.matrix(Tip ~ ., data = test_clean)
Y_test = test_clean$Tip

# Remove zero-variance columns from X_train
nonzero_var_cols = apply(X_train, 2, var) != 0
X_train = X_train[, nonzero_var_cols]

# Align X_test to match X_train columns
X_test = X_test[, colnames(X_train), drop = FALSE]

# Run regression
```

```r
source("mlr.R")
mlr_result = mlr(X_train, Y_train)
bhat = mlr_result$bhat
sebhat = mlr_result$sebhat

# Predict and evaluate
Y_pred = as.vector(X_test %*% bhat)
rmse = sqrt(mean((Y_test - Y_pred)^2))
cat("RMSE on Test Set:", round(rmse, 4), "\n")
```

```
## RMSE on Test Set: 1.1045
```

1. The logic behind the modeling choice The goal was to predict the tip amount based on various order features. Multivariable linear regression is a natural fit here because the output is continuous (tip), and we want to quantify how each feature affect the tip, since mlr provides interpretable coefficients, it is a good fit for interpretation
2. How the input features are used (for both models) -Taco.Type and Taco.Size as dummy variables. -Toppings.Count, Price, and Delivery.Time as numeric predictors.
3. Why this method is appropriate for your data set (for both models) he response (tip) is numeric, fitting the assumptions of linear regression. MLR gives interpretable coefficients to show how much each feature affect the tip.
4. Explanation of how you split your data into training and testing for model validation (for both models) We use 80% of the data on training and test the model on the rest 20% of the data, and then we check the RMSE. Then we Refit model on full data to get predictions.
5. Calculation of a performance metric (of your choice) (for both models) RMSE on test set = 1.1045, which means the predicted tips are, on average, about $1.1045 off from the actual tips.
6. A closing section that chooses which model appears to work best according to your metric and why you think so. See comparative analysis section at the end of the paper.
7. An example of five individuals with different food testing preferences and the recommendation for which your model recommends to them. It is shown above.

Comparative analysis conclusion

```r
test_data$Tip = test_data$Tip....
test_data$Restaurant = test_data$Restaurant.Name
recommended_restaurants = c("Grande Tacos", "Taco Haven", "La Vida Taco", "Spicy Taco House", "The Taco
avg_tip_cluster = mean(test_data$Tip[test_data$Restaurant %in% recommended_restaurants])
avg_tip_test = mean(test_data$Tip)

cat("Average actual tip for recommended restaurants (K-Means):", round(avg_tip_cluster, 4), "\n")
```

```
## Average actual tip for recommended restaurants (K-Means): 1.7777
```

```r
cat("Average tip in test set:", round(avg_tip_test, 4), "\n")
```

```
## Average tip in test set: 1.6816
```

```r
cat("RMSE on test set (MLR):", round(rmse, 4), "\n")
```

```
## RMSE on test set (MLR): 1.1045
```

```r
#Compute average tip per cluster using TRAIN data
train_data$Tip = train_data$Tip....
cluster_avg_tip = aggregate(train_data$Tip, by = list(Cluster = train_data$Cluster), FUN = mean)
colnames(cluster_avg_tip) = c("Cluster", "AvgTip")

#Compute global tip average on test set
test_data$Tip = test_data$Tip....
global_test_avg_tip = mean(test_data$Tip)

#Merge test cluster assignment with train-based cluster tip averages
test_data$Cluster = apply(dfs_test, 1, function(row) {
  dists = apply(kmc$centers, 1, function(center) sum((row - center)^2))
  which.min(dists)
})
test_data = merge(test_data, cluster_avg_tip, by.x = "Cluster", by.y = "Cluster")

#Accuracy
kmeans_correct = sum(test_data$AvgTip > global_test_avg_tip)
kmeans_accuracy = kmeans_correct / nrow(test_data)
cat("K-Means recommendation accuracy (high-tip cluster):", round(kmeans_accuracy, 4), "\n")
```

```
## K-Means recommendation accuracy (high-tip cluster): 0.305
```

This indicates that 30.5% of the time, K-Means recommended a cluster whose average tip is higher than the typical tip in the test set — a good proxy that it recommends 'better' restaurants.

Both models offer valuable insights, but they serve slightly different purposes. The K-Means clustering model identifies groups of restaurants that, on average, receive higher tips — around 1.78 dollars, compared to the test set average of 1.68 dollars. This suggests it's effective at surfacing popular or well-liked spots.

Meanwhile, the MLR model aims to predict individual tip amounts and achieves a test RMSE of 1.10. While this is lower error compared to naive guessing, the error is still relatively large given the typical tip range, indicating room for improvement.

Overall, K-Means may be more useful when the goal is to recommend consistently well-tipped restaurants, where MLR is better at provides personalized predictions that can help estimate expected tipping behavior for a specific order. Therefore K-Means is the better model.