Predicting Rideshare Prices

Krishna Kalakkad, Laura McGann,
Ethan Waite, Ethan Zimbelman

The mission

Given information about a rideshare trip, can we predict the price using KNN?

Input data

Data collected from the **Uber & Lyft Cab Prices** dataset on Kaggle

Cab rides

- Service (Uber or Lyft) and cab type
- Price and surge multiplier
- **Source**, destination, and distance
- Day of week, **hour, hour half (1,2)**

Weather

- Temperature, pressure, humidity, clouds, rain, and wind
- **Location**
- Day of week, **hour, half hour (1,2)**

Joined records and building datasets

Joined **rides** with **weather** on (day, hour, halfHour, location)

RDD Schema:

```
(day, hour, halfHour, location), LabeledRecord(id,
Record(distance, cab type, destination, source,
surgeMult, temp, clouds, pressure, rain, humidity,
wind), price)
```

Joined records and building datasets

Joined **rides** with **weather** on (day, hour, halfHour, location)

RDD Schema:

```
(day, hour, halfHour, location), LabeledRecord(id,
Record(distance, cab type, destination, source,
surgeMult, temp, clouds, pressure, rain, humidity,
wind), price)
```

Joined records and building datasets

- .take() only 1% of total data
- Split 80/20% train/test
- test.cartesian(train)
  - Results in 6.1M rows of data
  - ~404MB
- Cartesian Product of 100% of the data would be >40GB

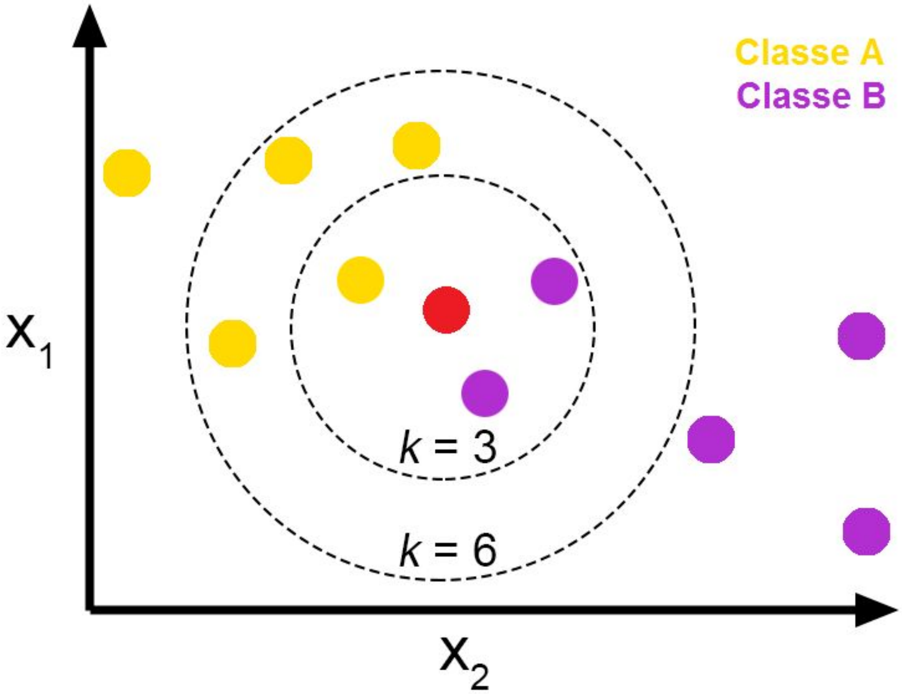
Standardizing Data

$$Z = \frac{x - \mu}{\sigma}$$

- val mean = scores.sum / count
- val devs = scores.map(score => (score - mean) * (score - mean))
- val stddev = Math.sqrt(devs.sum / count)
- return scores.map(x => (x - mean)/stddev)

KNN Overview

- ML classification model
  - Numerical or categorical
- "Training" = get training dataset
- For each test record:
  - Find k-nearest neighbors (distance)
  - Aggregate k-nearest's labels (avg)



Jose, Italo. "KNN Classification Example: Colored Scatter Plot with a Test Point and Different k-Sized Neighborhoods." *Towards Data Science*, 18 Nov. 2018, https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d. Accessed 2022.

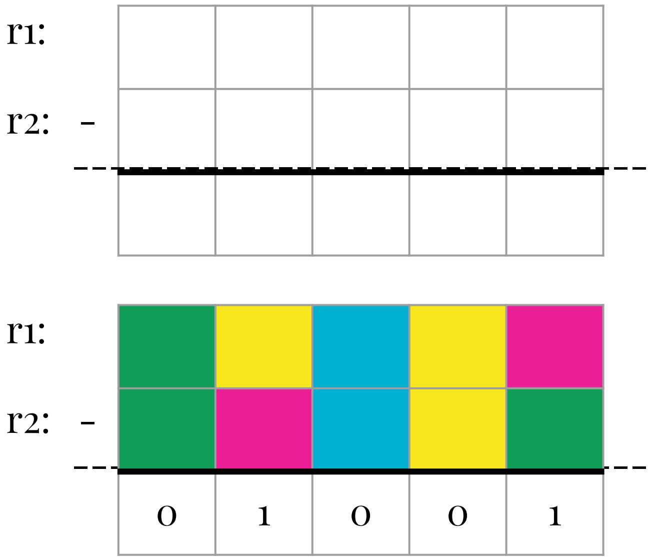
Distance matrix

- Euclidean distance for numeric attributes
  sqrt(sum((r1Numerical_i - r2Numerical_i)^2))
- Anti-dice distance for categorical attributes

$$\frac{\text{(total \# mismatches)}}{\text{(total \# categorical attributes in one record)}}$$

- Combine distances via weighted sum

$$\frac{\text{(\# numer)}}{\text{(total \# attrbs)}} * \text{numerDist} + \frac{\text{(\# cat)}}{\text{(total \# attrbs)}} * \text{catDist}$$

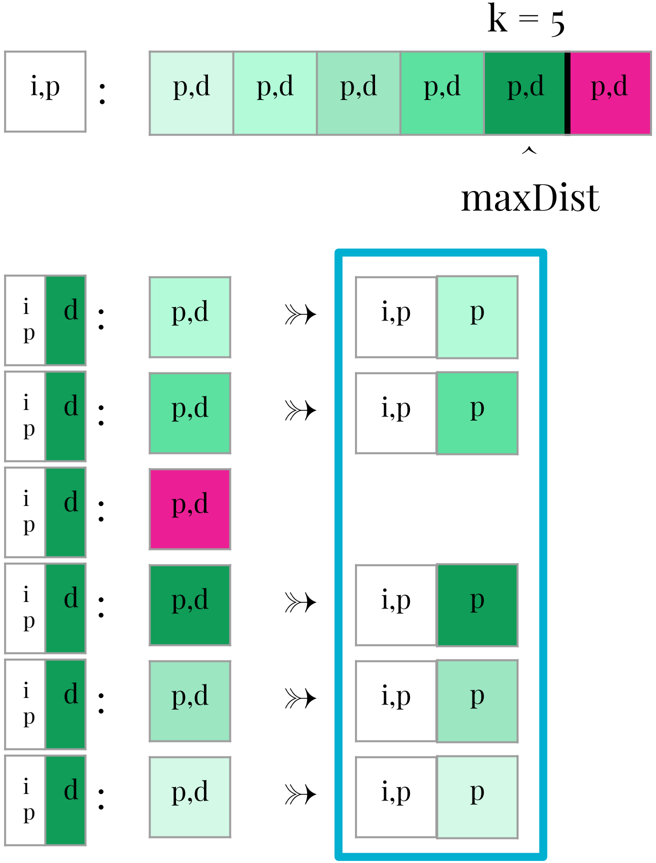


$$\boxed{\frac{5}{10} * \text{numerDist} + \frac{5}{10} * \frac{2}{5}}$$

K-Nearest Neighbors

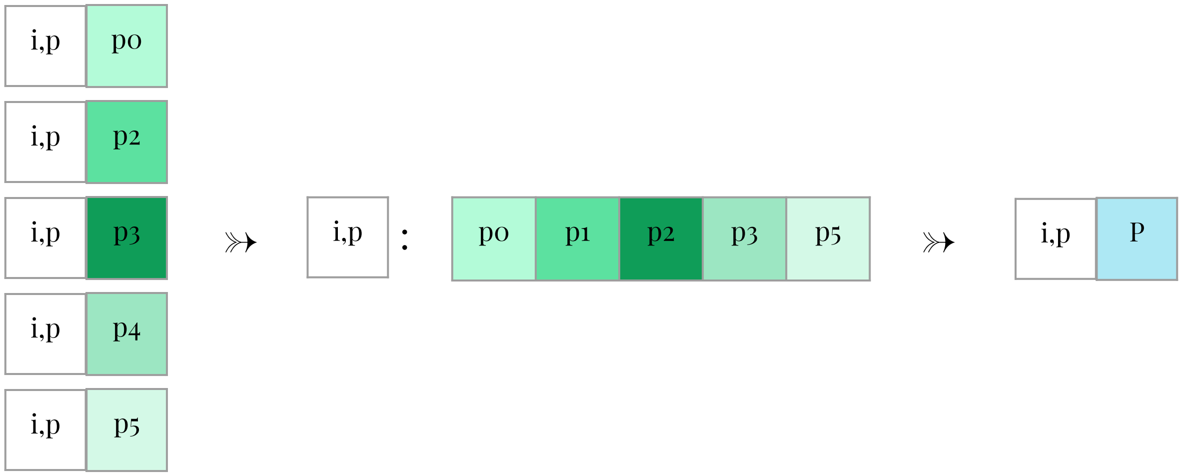
- topByKey(k)(dists descending)
- Take maxDist

- Join all train records <= maxDist away
- Resulting records:
  ((test_ID, test_price), train_price)

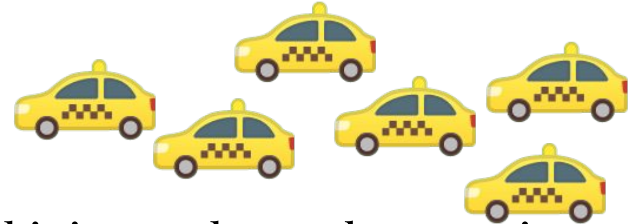


Predicting Price

- Aggregate prices for all K-nearest neighbors for each test record
- combineByKey()
  - Avg of all K-nearest prices = predicted price



Major obstacle: Lots of data



Performing cartesian products and joins on large datasets is **expensive!!!!**

- A cartesian product with ~500k testing records and ~120k training records creates **60,000,000,000 records!** That's a lot!
  - Decided to work with a subset of data (1-2% of all records)

Amount of used records (%)	Time to compute distance matrix (ms)
(4956 train, 1239 test); 1% of all records	74,157ms
(9912 train, 2478 test); 2% of all records	296,577ms
(14868 train, 3717 test); 3% of all records	*ERROR (shuffle failure)*
(19824 train, 4957 test); 4% of all records	*ERROR (shuffle failure)*

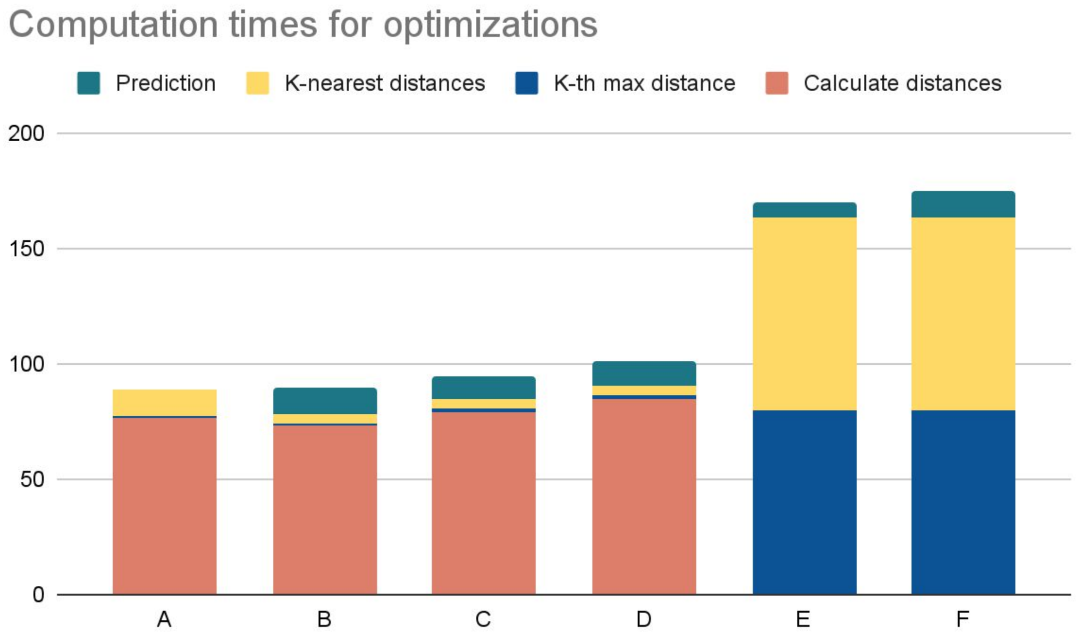
Time to compute the distance matrix by record sizes.

!!!!

The 4x computation time increase for a 2x record count increase is a result of performing a cartesian product

Optimizations: Scala

- Built in functions
- Partitioning data
- Removing persist()



Future optimizations

- Reducing object size (attributes and attribute sizes)

Optimizations: KNN

Hyper-parameterizations

- Train/test set size
- K-value

	50% train/ 50% test	60% train/ 40% test	70% train/ 30% test	80% train/ 20% test	90% train/ 10% test
K = 5	$6.59	$6.24	$5.87	$5.82	**$5.37**
K = 10	$6.90	$6.65	$6.56	$6.69	$6.53
K = 50	$7.17	$6.93	$6.96	$7.01	$6.97
K = 100	$7.30	$7.06	$7.07	$7.15	$7.06

The average errors of different parameters of the KNN algorithm on the same dataset.

Future improvements

- Modifying the attributes considered by the distance function
- Modifying the data?

Key learnings

- Cartesians are costly. Joins are too.

- Be persistent! don't persist() single use RDDs
- Partition data used in joins!
- It's not about the destination, it's about the journey ✨

Public transit pricing can be predicted **much more accurately!**

Questions?