DSCI 617: BIG DATA FINAL PROJECT

Lauren Schmiedeler

PROJECT STEPS

DATA PREPARATION

Read Data, Remove NAs, Assemble Features Using Vector Assembler, Create Training and Test Sets

MODELING AND PREDICTION

Linear Regression, Decision Tree, Random Forest, Gradient Boosted Tree, Prediction, Evaluation, Hyperparameter Tuning

DATA PREPARATION

Assemble Features Using Vector Assembler, Scale Features Using Min Max Scaler

CLUSTERING

K-Means, Evaluation

DATA PREPARATION: OVERVIEW

This dataset is from Kaggle and includes taxi trips for 2016, reported to the City of Chicago in its role as a regulatory agency.

Descriptions of Relevant Features:

- **trip_seconds:** time of the trip in seconds
- **trip_miles:** distance of the trip in miles
- **fare:** the fare for the trip

```
number of rows = 19866157
```

taxi_id	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_census_tract	dropoff_census_tract	pickup_community_area	dropoff_community_area	fare
85	2016-01-13 06:15:00	2016-01-13 06:15:00	180	0.4	null	null	24	24	4.5
2776	2016-01-22 09:30:00	2016-01-22 09:45:00	240	0.7	null	null	null	null	4.45
3168	2016-01-31 21:30:00	2016-01-31 21:30:00	0	0.0	null	null	null	null	42.75
4237	2016-01-23 17:30:00	2016-01-23 17:30:00	480	1.1	null	null	6	6	7.0
5710	2016-01-14 05:45:00	2016-01-14 06:00:00	480	2.71	null	null	32	null	10.25
1987	2016-01-08 18:15:00	2016-01-08 18:45:00	1080	6.2	null	null	8	3	17.75
4986	2016-01-14 04:30:00	2016-01-14 05:00:00	1500	18.4	null	null	null	null	45.0
6400	2016-01-26 04:15:00	2016-01-26 04:15:00	60	0.2	null	null	16	16	3.75
7418	2016-01-22 11:30:00	2016-01-22 11:45:00	180	0.0	null	504	8	32	5.0
6450	2016-01-07 21:15:00	2016-01-07 21:15:00	0	0.0	null	null	null	null	3.25

DATA PREPARATION: STEPS

- Create a list containing the file names of the monthly csv files using **os.listdir()**.
- Read in the monthly csv files using **spark.read.csv()**.
- Select only the necessary columns (**trip_seconds**, **trip_miles**, and **fare**).
- Remove the rows that contain **NAs** in any of the three remaining columns.

```
number of rows = 19862606
```

- Assemble the **trip_seconds** and **trip_miles** columns into a **features** column using **VectorAssembler()**.
- Rename the **fare** column **label**.
- Create a **training set** that contains 70% of the data and a **test set** that contains the remaining 30% of the data (seed = 100).

```
number of observations in the training set = 13902282
number of observations in the test set = 5960324
```

DATA PREPARATION: SUMMARY OF DATA (BEFORE TRAIN/TEST SPLIT)

df.show(10)

label	trip_seconds	trip_miles	features
4.5	180	0.4	[180.0,0.4]
4.45	240	0.7	[240.0,0.7]
42.75	0	0.0	(2,[],[])
7.0	480	1.1	[480.0,1.1]
10.25	480	2.71	[480.0,2.71]
17.75	1080	6.2	[1080.0,6.2]
45.0	1500	18.4	[1500.0,18.4]
3.75	60	0.2	[60.0,0.2]
5.0	180	0.0	[180.0,0.0]
3.25	0	0.0	(2,[],[])

df.summary.show()

summary	fare	trip_seconds	trip_miles
count	19862606	19862606	19862606
mean	13.892086664760871	767.0163579240307	3.394684370219944
stddev	25.385934033731534	1060.416563035996	22.597176756854346
min	0.0	0	0.0
25%	6.25	300	0.1
50%	8.5	540	1.1
75%	14.25	900	2.7
max	9999.0	86399	3353.1

MODELING AND PREDICTION: STEPS

- Using the **training data**:
  - Build a **linear regression** model with elastic net regularizers using **LinearRegresion(elasticNetParam = 0.5)**.
  - Build a **decision tree** model using **DecisionTreeRegressor(seed = 100)**.
  - Build a **random forest** model using **RandomForestRegressor(seed = 100)**.
  - Build a **gradient-boosted tree** model using **GBTRegression(seed = 100)**.
- Using the **test data**, make **predictions** for each model.
- **Evaluate** the models using the predictions.
  - Find **RMSE** and $\mathbf{R^2}$.
- Select the best model.
- Perform **hyperparameter tuning** on the best model using **CrossValidator(seed = 100)**.

MODELING AND PREDICTION: EVALUATION

model	RMSE	R2
GBTRegressor	23.25334	0.19844
RandomForestRegressor	23.35318	0.19154
DecisionTreeRegressor	23.35943	0.19111
LinearRegression	24.86094	0.08378

- The **linear regression** model is slightly worse than the other models based on RMSE and quite a bit worse based on R^2.
- The other three models perform similarly with the **gradient-boosted tree** model performing slightly better than the **decision tree** and **random forest** models.
- However, the **decision tree** model is **simpler** than the other two.

MODELING AND PREDICTION: HYPERPARAMETER TUNING

DecisionTreeRegressor(seed = 100)

CrossValidator(numFolds = 3, seed = 100)

Possible Parameters:

- minInstancesPerNode = [1, 2, 4]
- maxDepth = [5, 10]

Best Parameters:

- minInstancesPerNode = 4
- maxDepth = 10

RMSE = 23.24027

R^2 = 0.19934

label	prediction
9500.45	40.2066219178905
9476.21	57.00532193803531
9300.45	35.75285125770721
9276.62	57.00532193803531
9026.31	26.85222517245868
9002.29	37.87905074626866
9001.52	15.904370564212362
9001.17	9.123955842988858
9001.0	12.069101031367095
9000.62	7.2128795830498325

For large labels, the predictions are very far off from the true values.

DATA PREPARATION: STEPS

- Consider only the **training data**.
- Assemble the **trip_seconds**, **trip_miles**, and **fare (label)** columns into a **k_means_features** column using **VectorAssembler()**.
- Scale the features and create a **k_means_features_scaled** column using **MinMaxScaler(min = 0, max = 1)**.
  - Without scaling, only the larger features will affect the k-means clustering.
  - In this situation, **trip_seconds** (mean = 767) would completely outweigh **fare (label)** (mean = 13.9) and especially **trip_miles** (mean = 3.4).
  - After scaling, the values for each feature are between 0 and 1.
- Select only the columns relevant to k-means clustering (all the columns except **features**).

DATA PREPARATION: SUMMARY

label	trip_seconds	trip_miles	k_means_features	k_means_features_scaled
9999.0	12	0.0	[12.0,0.0,9999.0]	[1.388904964177826E-4,0.0,1.0]
9890.12	11820	161.6	[11820.0,161.6,9890.12]	[0.13680713897151586,0.048194208344517014,0.9891109110911093]
9800.45	1980	0.0	[1980.0,0.0,9800.45]	[0.02291693190893413,0.0,0.9801430143014302]
9739.58	1140	716.0	[1140.0,716.0,9739.58]	[0.013194597159689347,0.2135337448927858,0.974055405540554]
9600.48	3780	0.0	[3780.0,0.0,9600.48]	[0.04375050637160152,0.0,0.9601440144014401]
9500.45	2220	0.0	[2220.0,0.0,9500.45]	[0.025694741837289783,0.0,0.9501400140014002]
9490.61	300	91.8	[300.0,91.8,9490.61]	[0.0034722624104445653,0.027377650532343208,0.9491559155915592]
9400.24	960	0.0	[960.0,0.0,9400.24]	[0.01111123971342261,0.0,0.9401180118011802]
9200.46	3240	0.0	[3240.0,0.0,9200.46]	[0.0375004340328013,0.0,0.92013801380138]
9200.45	3600	0.0	[3600.0,0.0,9200.45]	[0.04166714892533478,0.0,0.9201370137013702]

CLUSTERING: STEPS

- Consider only the **training data**.
- Create ten different **k-means** models, each using a different value of k.
  - Try k = 2, 3, 4, ..., 11.
- **Evaluate** the k-means models.
  - Find the **silhouette value** for each model.
- Select the **best value of k** using the **silhouette values**.
  - Consider the large size of the data.

CLUSTERING: EVALUATION

- The best **silhouette value** is associated with **k = 2**, and the second-best **silhouette value** is associated with **k = 5**.
- Considering the size of the data (the training data contains almost 14 million observations), I would use **k = 5** instead of **k = 2**.
- Splitting a dataset this large into **5 clusters** seems more reasonable than splitting it into only **2 clusters**.

k	silhouette
2	0.998739393565863
3	0.7994544107024001
4	0.7236816173458225
5	0.8400285858935831
6	0.7468181144931019
7	0.6423728508376305
8	0.7398369725001089
9	0.6827657265896455
10	0.6249826754905856
11	0.6504570983637393