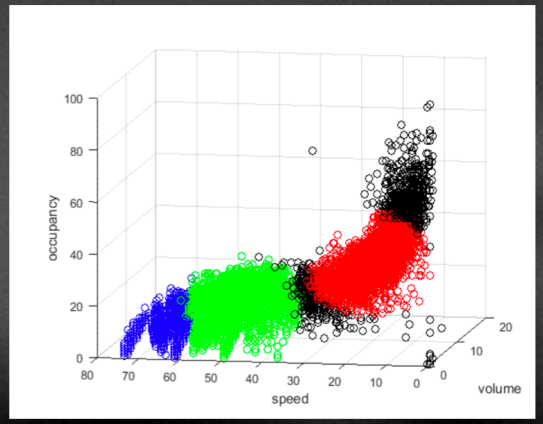
Triple Y presents

Detecting Null and Outliers

Group Member: Yulin Shen, Yitao Liu,Yiyang Sun

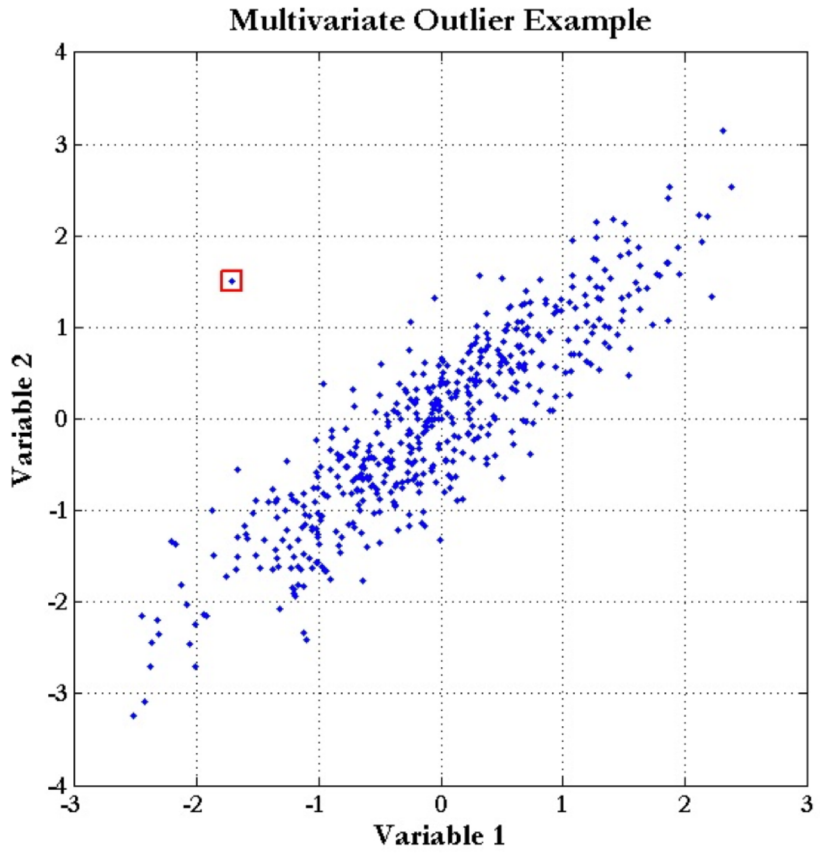
OVERVIEW:

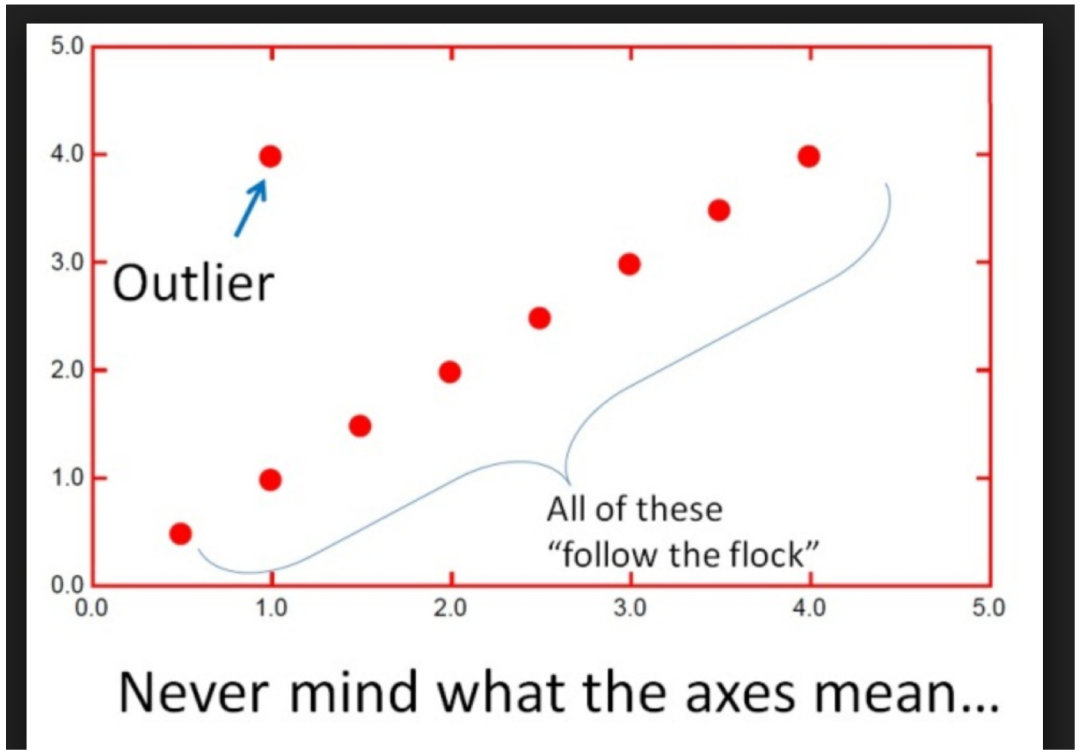
- What is null and outlier
- Existing algorithm and approach
- Our approach: K-means + AVF
- Results and Strengths



Introduction: null and outliers

What is an outlier?





Outliers are instances which do not conform to other items in a dataset

Introduction: null and outliers

Outliers are not useless!

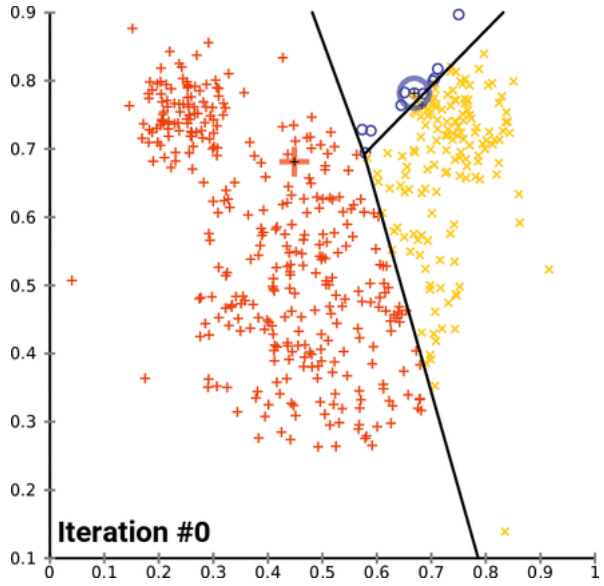
Some may be extremely useful. For example: credit card fraud detection
Outliers are always fraud usages.

Existing Approach

K-MEANS

- Unsupervised clustering
- Set number of clusters manually
- Minimize the within-cluster sum of squares to each cluster's center point(Euclidean distance)
- Time-complexity: O(n^2)
- K largest distances as outliers

How k-means works



Existing Approach

Problem of K-Means

Euclidean distances are only capable for numerical data!

How to deal with categorical data?

AVF algorithm

Existing Approach

Attribute Value Frequency AVF

- Linearly scalable

- Calculate the frequency of each single attribute in a column

- Sum up all frequencies in the rows

- K-lowest scores as outliers

Example:

attribute 1	attritube 2	a1 score	a2 score	avf score
a	b	3	1	2
c	c	1	1	1
a	a	3	2	2.5
a	a	3	2	2.5

Existing Approach

Problems of AVF

If numerical columns are much more than categorical columns, AVF only use small subset of columns.

Solution? Our Approachs!

Our Approach

Pre-processing:

- Clear out all the nulls (set a threshold), for example: if we set the threshold to be 50%, then if more than 50% of the values in a row are null, then we define it as a null row and clear it out

```
def find_null(x):
    col_ind = x[-1]
    count = 0
    for i in range(len(x)-1):
        if x[i] == "NA" or x[i] == "":
            count += 1
    if count >= len(x)/2:
        x[-1] = (col_ind,-1)
    return x
```

Our Approach

Pre-processing:

- Automated detection of numerical or categorical columns

```
for n in range(len(tsv_rdd.first())-1):
    num_check = 0
    test_col = tsv_rdd.map(lambda x:x[n])
    test_col_take = test_col.take(test_count)
    #num_uniq = len(np.unique(test_col))
    num_uniq = test_col.distinct().count()
    if num_uniq < 51:
        Cat_Cols.append(n)
    else:
        for m in range(test_count):
            try:
                float(test_col_take[m])
            except:
                num_check = num_check + 1
        if num_check < 0.2*test_count:
            Num_Cols.append(n)
```

Our Approach

Pre-processing:

- Clean dirty data (for example, “13a” appears in a column classified as numerical)

```
def find_dirty(x):
    col_ind = x[-1]
    for i in Num_Cols:
        try:
            float(x[i])
        except:
            x[-1] = (col_ind,-1)
    return x
```

```
tsv_rdd = tsv_rdd.map(lambda x: find_dirty(x))
dirty_row = tsv_rdd.filter(lambda x: type(x[-1]) != int).map(lambda x: str(x[-1][0]))
tsv_rdd = tsv_rdd.filter(lambda x: type(x[-1]) == int)
```

Our Approach

Improved K-Means

- Use the concept of AVF to convert all categorical data to their frequencies, thus make it as numerical data.

```
def string_freq(x):
    for e in Cat_Cols:
        #in dict: {x[e]:freq of x[e]}
        x[e] = str(dict_lst[ind_dict[e]][x[e]])
    return x
```

```
if Cat_Cols != []:
    #dict_lst=[dict1_for_col1, dict2_for_col2, ...]
    dict_lst = []
    for i in Cat_Cols:
        dict_lst.append({})
    #ind_dict = {col_nbr:position_in_dict_list}
    ind_dict = {}
    dict_i = 0
    for e in Cat_Cols:
        new_coldict(dict_i, e, total_lines) #add entries in cols to d
        ind_dict[e] = dict_i #record col_nbr and dict_lst position
        dict_i += 1
    tsv_rdd = tsv_rdd.map(lambda x: string_freq(x)) #change string to
```

Our Approach

Improved K-Means

- Standardize every column so that each column will be equally weighted.

```
#Kmeans Cols
a = tsv_rdd.map(lambda x: np.array(gen_lst(x)))

#normalization
scaler1 = StandardScaler().fit(a)
a = scaler1.transform(a)

#Kmeans
```

Our Approach

Improved K-Means

Main program:

- Set number of clusters, and run K-means on the dataset
- Set the number of outliers, and emit the points that have the farthest distance

Our Approach based on AVF

Improvement:

Main program:

Find maximum and minimum

Set 50 buckets by the difference

Convert numerical column into

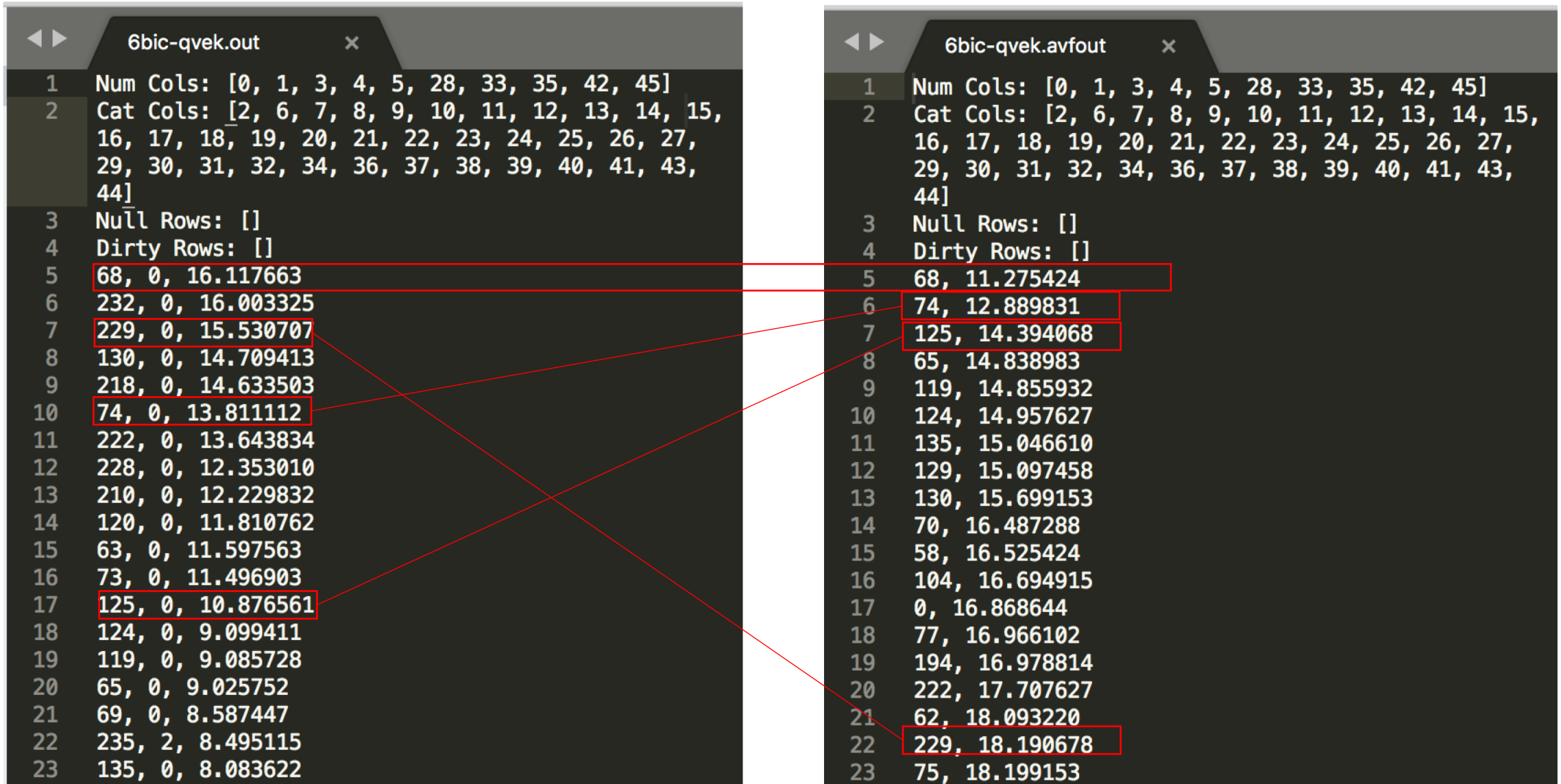
a 50-class categorical column

```
def bucket_value(x):
    if current_bucket == 0:
        x[current_col] = "-1"
    else:
        tempval = (float(x[k]) - bucket_min)//current_bucket
        x[current_col] = str(tempval)
    return x
```

```
current_col = 0
current_bucket = 0
bucket_min = 0
for k in Num_Cols:
    test_col = tsv_rdd.map(lambda x: float(x[k]))
    tsv_rdd.map(lambda x: float(x[k])).max()
    diff = float(test_col.max()) - float(test_col.min())
    bucket = diff/50
    current_col = k
    current_bucket = bucket
    bucket_min = float(test_col.min())
    tsv_rdd = tsv_rdd.map(lambda x: bucket_value(x))
```

Our Approach:

Results: Improved K-Means **VS** Improved AVF



6bic-qvek.out

```
1  Num Cols: [0, 1, 3, 4, 5, 28, 33, 35, 42, 45]
2  Cat Cols: [2, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
   16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
   29, 30, 31, 32, 34, 36, 37, 38, 39, 40, 41, 43,
   44]
3  Null Rows: []
4  Dirty Rows: []
5  68, 0, 16.117663
6  232, 0, 16.003325
7  229, 0, 15.530707
8  130, 0, 14.709413
9  218, 0, 14.633503
10 74, 0, 13.811112
11 222, 0, 13.643834
12 228, 0, 12.353010
13 210, 0, 12.229832
14 120, 0, 11.810762
15 63, 0, 11.597563
16 73, 0, 11.496903
17 125, 0, 10.876561
18 124, 0, 9.099411
19 119, 0, 9.085728
20 65, 0, 9.025752
21 69, 0, 8.587447
22 235, 2, 8.495115
23 135, 0, 8.083622
```

6bic-qvek.avfout

```
1  Num Cols: [0, 1, 3, 4, 5, 28, 33, 35, 42, 45]
2  Cat Cols: [2, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
   16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
   29, 30, 31, 32, 34, 36, 37, 38, 39, 40, 41, 43,
   44]
3  Null Rows: []
4  Dirty Rows: []
5  68, 11.275424
6  74, 12.889831
7  125, 14.394068
8  65, 14.838983
9  119, 14.855932
10 124, 14.957627
11 135, 15.046610
12 129, 15.097458
13 130, 15.699153
14 70, 16.487288
15 58, 16.525424
16 104, 16.694915
17 0, 16.868644
18 77, 16.966102
19 194, 16.978814
20 222, 17.707627
21 62, 18.093220
22 229, 18.190678
23 75, 18.199153
```

Our Approach

K-Means + AVF

Key Strength:

- Works for any types of dataset, no matter numerical or categorical or mixed.
- Spark RDD, able to deal with the large datasets

Thank You!