Instance selection in Big data

Siddhartha Saxena
siddsax@cse.iitk.ac.in

Dr Om Deshmukh
odeshmukh@yodlee.com

Deepak Patil
dpatil@yodlee.com

Abstract

Mining for a representative data from Big data to feed into machine learning models retains to be a difficult problem, this problem know as instance selection has been tackled in both data driven and hashing based methods, in our work we tackle this problem in a data driven manner via online clustering. We cluster some sample of data initially and then use it to classify new data points, shuffle the model using these points and go on. We explore the challenges related to it and provide solutions for them.

1. Introduction

A major problem with big data is that machine learning models can consume only a small amount of data, even after having large volumes of data, one can't feed them to models to improve generalizability. This leads to a central problem in Big Data Analytic, to learn a better and diverse representation of the data points. Overcoming this problem can tremendously boost the generalizability of the models built upon it as they'll have access to a wider variety of data points that can emulate the test data.

2. Defining the Problem

The problem at a lower level is to extract descriptions that form an archetype. It is a data-driven data sampling method as opposed to a rule-based method. The obvious advantage of this over the other method is that it learns the features required on its own and can evolve over time on its own which is lacking in the former, but the biggest problem with applying complex machine learning models is that they can't scale to such large data sizes.

The task involves two parts, first is to create vectors from data, as any machine learning model needs numbers to interact with data, and the other part is sampling from them. These two parts are explained in the next sections.

3. Finding features of words

Having good feature vectors is an important step for preparing the model for sampling. Below are the techniques considered by us in this work.

3.1. Bag of Words Model or n-gram model

The most basic method to represent sentences is the bag of words model where we form a dictionary of words or n-grams in the case of an n-gram model, and then create a vector with counts of each dictionary word in the description, but this faces several critical issues. First of all, this doesn't capture the semantic meanings between different words in the dictionary. Apart from this, it also deals with very dimensional feature vectors as the size of vectors increase linearly with the size of the dictionary and finally as most of the vectors are zeros, it provides sparse vectors which are highly problematic for machine learning models.

3.2. Word2Vec

Word2Vec as proposed by Mikolov *et al.*[5] provides a much more suitable model to represent sentences or in our case descriptions in vectors of fixed dimensions. It learns a vector corresponding to each word and then averages the vector for each word in a description to represent it. Thus it is able to capture the semantic meaning of the sentence along with solving the problem of very high dimensionality. To learn the encodings the word2vec model uses the context of the given word, i.e. it optimizes the probability of the given word to occur when the words near it are given as input to the model as one-hot encodings.

3.3. Sent2Vec

A new method for the task of learning embedding for a small paragraph or sentence is the Sent2Vec model[6], it another highly efficient model that outperforms previous unsupervised word embedding models including word2vec, it even outperforms some of the supervised word embedding models on some task. It uses the whole sentence as the window for getting context and is especially tuned for the same, hence it can be exported for use in future works. The code for the same is available **here(Link)**

3.4. fastText

fastText [4] goes a level further than word2vec and uses character n-grams instead of words to produce feature vec-

tors, hence it can even learn vectors for unknown words, i.e. not present in the dictionary. This can tremendously help feature learning due to the long tail present in the word distribution of words. The code for the same is available **here(Link)**

4. Performance of word2vec on descriptions

We used K-Nearest Neighbours to test word2vec. We used a 30K dataset to test with a dictionary size of 20K, the qualitative output is as summarized in Table 1.

As showed by the Table 1, the nearby descriptions are the ones that are having similar words but it is still better than a simple word dictionary model as it alleviates the problem of sparse vectors and fixed dimension, it is a simple and effective solution to the problem at hand, although the performance can be further improved using the methods described in section 3.3 and 3.4.

5. Instance selection

Instance selection is a central problem in big data analytics as explained above, and there has been work upon the same by researchers. The approaches considered by us are underlined as follows

5.1. Traditional Instance Selection Methods

Work is being doe on this problem for a long time, but most of it is done in a supervised setting, especially by building upon the simple K-Nearest Neighbour Classification. It has not been a hot field of research in academia even after tremendous requirement in the industry as academics don't have to deal with a large amount of data. Hence the models that are after all available like [2] [1]. [4]are not reliable. This leads us to come up with a new way to subsample a smaller dataset in an unsupervised setting.

5.2. Gaussian Mixture Model

we propose To cluster the data and then sample from all of the clusters with respect to their sizes. This will lead to a wider variety of samples because all clusters are varied from each other and have and each of them will have a certain specialty, this means they will span the whole data space and thus we can get varied samples as required.

The Gaussian Mixture Model is one of the best models for the task of clustering. The basic assumption in this model is that the data points come from a mixture of Gaussians, hence they have the flexibility of being of various sizes and shapes. It optimizes the means and covariance matrices of the cluster using the Expectation Maximization Algorithm which is quite fast but as we observed, it is not suitable for a large amount of data. It takes in a longer time to converge hence not worthy when compared to K-Means.

5.3. Kmeans with Kmeans|| initialization

Kmeans is one of the fastest and simplest clustering algorithms. Apart from that, it also gives very good results. It holds the assumption that the data comes from spherical clusters or hyperspheres in higher dimensions. This is a strong assumption which makes it lesser than Gaussian Mixture Model, but due to its speed, it is ideal for our purpose of clustering on Big Data. We also used Kmeans|| initialization in this work[3], Kmeans|| or scalable Kmeans++ initializes the centroids by iterating in K loops, in each iteration choosing a centroid with probability proportional to its distance from the closest point in the set of centroids.

5.4. Fast Approximate K-Means via Cluster Closures

In our work we also experimented with using Cluster Closures in K-means [7]. The idea behind it is to update only the cluster assignments of points that are at the boundaries of cluster closures, as they have a higher probability of having a change in cluster assignment. This decreases the time required for the assignment step, but we found out that the major time was being taken in the aveaging and counting step for each cluster which is a necessary step for K-means, hence we had to abandon that approach.

6. Work-flow

Take data from any source and extract

Listing 1. Part 1

```
cut -d'|' -f{column number} {Data file}\
> data.txt
python count.py data.txt
python tail.py data.txt {number of\
final samples}
python3 word2vec_PICKLE2GPU.py data.txt
#saves the trained word encodings as
# word2vec_dict.npy and outputs
#desc_embeddings.npz as embeddings
```

In case any error due to encodings while reading file use iconv -f us-ascii -t utf-8 Input file -c tb > Output File

Word2Vec.py by default take all the words present in the description file as the dictionary, and as it is a simple (complexity wise) model, it can easily scale to a large dictionary size. If one needs custom dictionary size then just pass it as the final argument.

Another important point is that running Word2Vec on CPU is more beneficial than on GPU as CPU has more RAM than the GPU, thus it can hold a tensor of much larger sizes than possible by GPU. Hence after training word2vec model, the vectors are stored in the file word2vec_dict.npy as a dictionary, hence for getting vectors for new descriptions, the file word2vec_vectorize.py can be used with the

Description	Closest Neighbour	2nd Closest Neighbour	3rd Closest Neighbour
purchase bray and scarff tyson vienna va	return bray and scarff tyson vienna va	purchase le meridien delfina a sa santa monica c	purchase sq *talk n fix broomfield co
pending* wendys # nashville us wendys # nashville us	pending* tenn nashville us tenn nashville us	pending* raceway nashville us raceway nashville us	pending* mapcoexpress # nashville us mapcoexpress # nashville us
purchase mp flushing llc flushing ny	purchase marshalls # flushing ny	purchase mapo bbq flushing ny	purchase bj wholesale # flushing ny

Table 1. Qualitative results of Word2Vec outputs

1st argument as the file to vectorize and second as the dictionary. Steps after this need to be done on an Amazon EMR cluster or Spark should be installed.

An important step before running any code in 2nd part is to do one edit in a hadoop file open /etc/hadoop/config/capacity-scheduler.xml, then replace <value>org.apache.hadoop.yarn.util.resource.DefaultResourceCalculator</value> in line 36 with <value>org.apache.hadoop.yarn.util.resource.DominantResourceCalculator</value> otherwise only one core would be utilized as opposed to the number available.

Listing 2. Part 2

```
spark-submit  --driver-memory\
{num}g np2HDFS.py desc_embeddings.npz
# creates a folder of the name
#HDFSdesc_embeddings.npz which is the
#data in distributed format
spark-submit --driver-memory {num}g \
kmeans-save.py HDFSdesc_embeddings.npz \
{Number of clusters} { Max no of
iterations} #saves the clustering
#model as Final_Output_Clusters/
python sampling-kmeans.py HDFSdesc_\
embeddings.npz/Final_Output_Clusters/
```

In the above section, when dealing with large data then it's recommended to use large memory clusters, for example in the step involving creation of HDFS, memory is consumed by the host or driver machine hence it needs more memory, the executor memory is maximized by Amazon at the time of creation of cluster via the config file hence changing that is no necessary.

As the code may take a large time to run, it is useful to use screen first and then run it

6.1. Tools for Analyzing

For analyzing the accuracy of clusters, K-Nearest Neighbours is a useful algorithm. One can do the analysis done in Table 3 using it, which can show if clustering has been done properly or not. Besides, it is also useful for checking word2vec. One can check if similar words are actually near to each other or not.

```
python KNN.py {file with vectors} \
{number of neighbours/K} {index of data \
point whose similarity is required} \
{options - 'a'/'b'} {2nd index/file\
with descriptions}
```

KNN.py has 2 modes, mode 'a' (i.e. the fourth argument is 'a') is when we need to check what is the neighbour rank between two given data points and mode 'b' to display the next K neighbours for a given data point

```
python np2csv.py # to covert numpy
#array description vectors to a csv file
```

7. Problems Faced

7.1. Finding Optimal Clusters

Finding optimal clusters is a problem always faced in clustering, we used the standard approach of finding the "knee-jerk" point in the graph of the Intra-cluster distance and number of clusters. The intra-cluster distance is the distance of each point from its centroid, which indicates how good clustering is. We experimented with 30K data points and the graph is as plotted. Approximating by it, we get the ratio of **0.00057**. Not being an exact measure of the number of the clusters but it provides a benchmark for clustering on a larger dataset on which we can't afford to run multiple times. Although a better way is to take as many clusters as possible, as this will lead to clusters being empty in case the "ideal" number of clusters is less.

7.2. Time Complexity Analysis

For a single iteration in K-means, the time is $O(n^2*k)$, same is for GMM but off by some constant factor. Although as it is an iterative algorithm, the number of iteration increases with the number of data points and this leads to a much more severe time complexity, closer to exponential time complexity, hence a reasonable size of 15 Million is used for the trade-off between data and time taken (and more importantly Costs).

7.3. Costs

Cost is an important factor for considering Amazon AWS instances. The two broad variety of instances that are

Model	Data size	Number of Cluster	Time per iteration	Machine	Cost
KMeans	15M	10K	7 Minutes	1 M4.16xlarge	$3.2 per Hour
KMeans	15M	10K	12 Minutes	5 M4.4xlarge	$0.8*5 per Hour

Table 2. Time taken by different machines

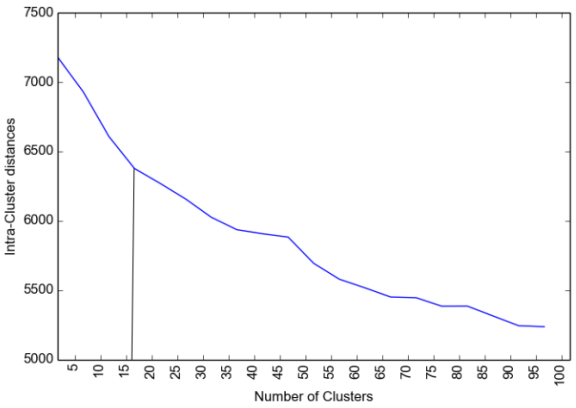

Figure 1. Graph of intra cluster distances and number of clusters showing the biggest knee-jerk around 17

available are the Amazon P2 instances and Amazon EMR. EMR instances are ideal for any spark related code whereas to run word2vec P2 instance is required (it is having a graphic card). Word2vec is just a shallow neural network or one can say it involves a matrix multiplication, hence it is very fast and efficient. Thus it can be easily done in a small instance like P2.xlarge. On the other hand for a large amount of data, clustering has high time and space complexity and to yield the maximum benefit of spark, one need to use as many cores as possible. Amazon EMRs are ideal for the same and have pre-configured master and slave nodes as well as offer spark-GUI to track the running jobs.

Table 2 shows the performance of different EMR cluster configuration on 10 Million data points with 10K clusters.

7.4. Space and Memory

As parts of this work are being done in two separate machines, due to their respective advantages, space and memory become important considerations. The biggest problem due to this issue is that it is necessary to store the word2vec embeddings in a file and then load it directly in pyspark as an RDD. This calls for a machine with large RAM storage as it is required to store numpy arrays and deserialize it to convert in an RDD to store it. Storing as a Pickle file which can be directly loaded as an RDD is not an option due to it being highly memory inefficient (RAM) whereas one can't store it directly as a CSV as a CSV file can become tremendously big and tougher to handle. Hence the solution opted was tp store it as a numpy array and then load it in an EMR machine with large RAM like m4.16xlarge and store as an RDD. Thus it can later be loaded into any machine, even with smaller RAM as it doesn't take much space in RAM when loaded as RDD.

7.5. Exponential distribution of words

Afer further analyzing the data, we observed that it was highly skewed. This meant that some words had very high frequency whereas most of the words were very less in number i.e. a long tail. To counter this problem, we used a different sampling method for the base of cluster data. Instead of randomly collecting data, we collected the data points in such a way that in the sampled data, the distribution of data has a fraction of the slope before, rejecting a description if it violates this fact. This leads to a lower slope in word frequency distribution which is highly beneficial for word2vec as it provides richer contexts for the descriptions consisting of words at the tail, whereas sufficient context is for the descriptions with very common words. This approach also leads to much more diverse clusters from the same data.

7.6. Tokenization

Some basic tokenization is also been done, that is to remove the duplicate descriptions which remove a large amount of descriptions from any data, then capitals are converted to small and the special characters are removed as well.

8. Online Clustering Model

Clustering or sampling directly from a very large as well as growing dataset is an impossible and misguided task. As data increases the time of clustering increased exponentially which makes it unfeasible, hence moving towards an online method was a natural choice. The method that we proposed is consist of the following steps.

1. Cluster some smaller sample of data created using rule-based clustering and then normalizing the number of words occurring as highlighted in section 7.5
2. Take more data and then classify them among the clusters using the model trained previously with a threshold, if a data point is out of that threshold then form a new cluster around it.
3. When the points are clustered then update the cluster centroids as the mean of the cluster members.

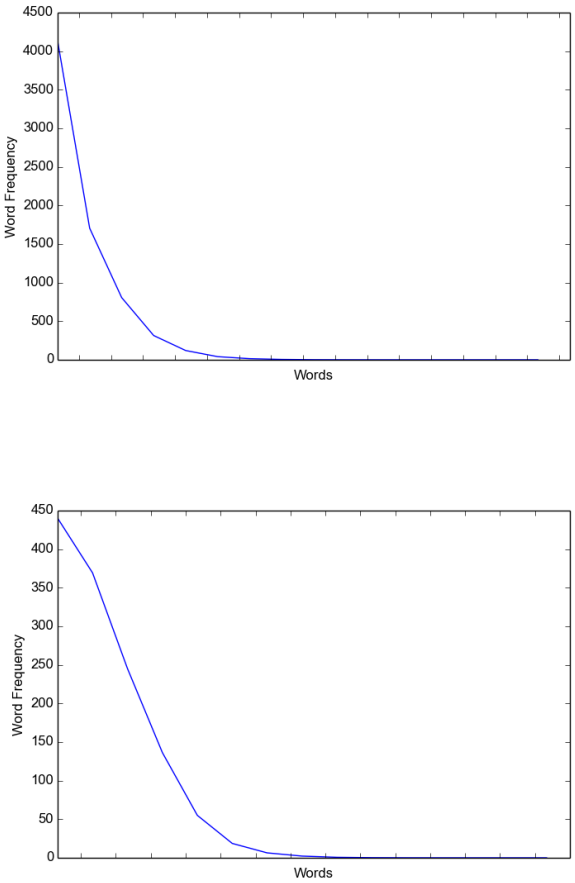

Figure 2. Exponential of word rank vs the number of times it occur. First is without sloping and second is with sloping

4. Repeat the process further by sub-sampling points from each cluster according to their sizes so that complexity of the problem is kept in check and repeat the process again.

This clustering model can scale to a large as well as streaming sample of data easily as the time taken at test time is much less than the time while training, further it is also memory efficient as we only need to store the learned parameters in memory.

9. Results

9.1. Analysis of Cluster outputs on 30K samples

Table 3 shows how the data points within the clusters are related to each other. The third column comprises of the data points closest to the centroid after the point in the second column, with increasing distance from top of the table to its end. The fourth column is the neighbour number of

Cluster No.	Data Point closest to centroid	Neighbour Number of data points next closest to Centroid
0	shell oil tenafly nj auto fuel dispenser	54 74 360 32
8	starbucks store new york ny qhqoitqk fast food restaurant	1 125 2 13
9	livestock tavern honolulu hi dusty@ breakingbreadhg	24 6 11 31

Cluster No.	Data Point closest to Centroid	Cluster IDs of 2 nearest Neighbours	Closeness rank wrt centroid
16	2129	16 16	25 7
13	7039	13 13	11 26

Table 3. Analysis of clustering on 10K dataset with K = 20

the corresponding data point wrt the data point in column 2. This illustrates that the point which are closer to the centroid are also close to each other which should happen ideally as well. Link to data

The first table shows that the points that near to the centroid are also close to each other, in case of the 1st cluster, the distance of all the points is a bit off from each other which can be explained as the centroid is the average of all the points hence the closest point from it might be a bit off, leading to other points being a bit far from each other.

9.2. Analyzing word frequency in clusters

Analyzing the clusters for word frequency reveals the identity of the clusters in terms of the words, clear patterns could be seen with a particular cluster consisting of description of a given word, hence upon sampling from these clusters one can get descriptions which are different from each other in terms of the words present in them which can provide a wider context to models built upon this data which is the final aim of our work.

9.3. Testing on Tagged data

We experimented with the tagged data to explore if there existed any patterns in cluster labels and tags present in a description. As shown in the graphs we found out that almost all the clusters had similar tags, this can be contributed to the fact that tags present in all the descriptions is very similar. Most of them have a vendor_name, city, pymnt_type (payment type).

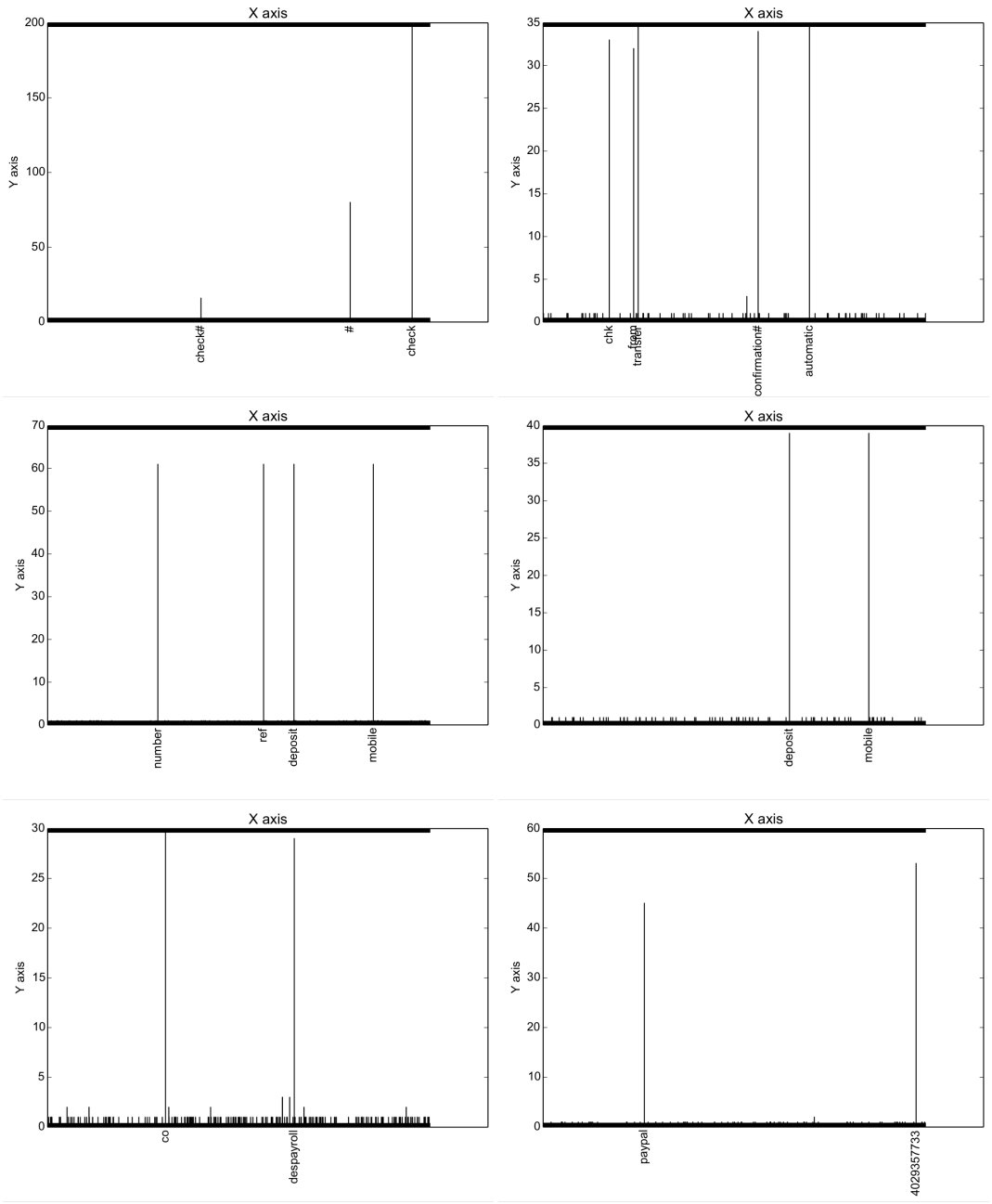

Figure 3. 4 clusters out of the 10K, the points within the clusters are more or less evenly spread with the maximum of them having .02 (on test data) as we can see, each cluster has a given pattern hence peaks in the word frequency

Cluster No.	Sample Descriptions	Cluster No.	Sample Descriptions
1	checkcard 0211 burger king #7025 check 00000000270	2	automatic transfer from chk 3317 ..# 1406609225 automatic transfer from chk 5452 ..# 1415934989
3	mobile deposit ref number 209090831891 mobile deposit ref number 210260930875	4	mobile deposit 45426460 xxxxx3311 mobile deposit 51333787 xxxxx9057
5	nation.. desnwide co27 id45.. indn.. co id5314 chase despension id00.. indn.. co id9164	6	paypal *ebayincship 4029357733 98.. paypal teacherspay 4029357733 05..

10. Best Practices and Notes Summarized

1. **Tokenization** Up till now only few things have been done that is removing special characters, converting to small and removing duplicates. This helps although a stronger tokenization like removing numbers, masks like xxxxx can help in getting better clusters as there are descriptions that are different from each other only due to some transaction numbers and form a cluster.

2. **Indexing** Code everywhere returns index numbers of lines rather than line numbers (as python indexes from 0) except for one place that is the NANs.txt given as output of word2vec which gives the line numbers of descriptions that need to be removed for tallying with them.

3. **EMR Cluster** It is better to use m4.4xlarge EMR instance when working on clustering data using a pretrained model. It has a good combination of RAM and number of cores, whereas a larger machine like m4.16xlarge for training.

4. **Flatter dataset** As highlighted in section 7.5, it is recommended to use a flatter data, although even more important is to use a dictionary of a large size, ideally containing all the words possible in a descriptions. This may seem like very complex but due to the simplicity of word2vec it is not very computationaly expensive, although it can be memory expensive but that can be overcome by running on CPU, which has a larger memory than GPU.

5. **Number of Clusters** It is beneficial to run clustering with as many clusters as possible but the running time of K-means increases a lot with increased number of clusters, hence we reached upon a sensible number of clusters, 10K for 15M data using the data we had for 30K dataset. This is a decent figure as it computationaly feasible as well as not very small, we ran 100 iterations of K-means of 15M data and it took around 20 hours on m4.16xlarge to run.

References

[1] A. Arnaiz-González, J. F. Díez-Pastor, J. J. Rodríguez, and C. García-Osorio. Instance selection of linear complexity for big data. *Knowledge-Based Systems*, 107:83 – 95, 2016.

[2] Á. Arnaiz-González, A. González-Rogel, J.-F. Díez Pastor, and C. López-Nozal. Mr-dis: democratic instance selection for big data by mapreduce. *Progress in Artificial Intelligence*, pages 1–9, 2017.

[3] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, Mar. 2012.

[4] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching Word Vectors with Subword Information. *ArXiv e-prints*, July 2016.

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *ArXiv e-prints*, Jan. 2013.

[6] M. Pagliardini, P. Gupta, and M. Jaggi. Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features. *ArXiv e-prints*, Mar. 2017.

[7] G. Zeng. Fast approximate k-means via cluster closures. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3037–3044, Washington, DC, USA, 2012. IEEE Computer Society.

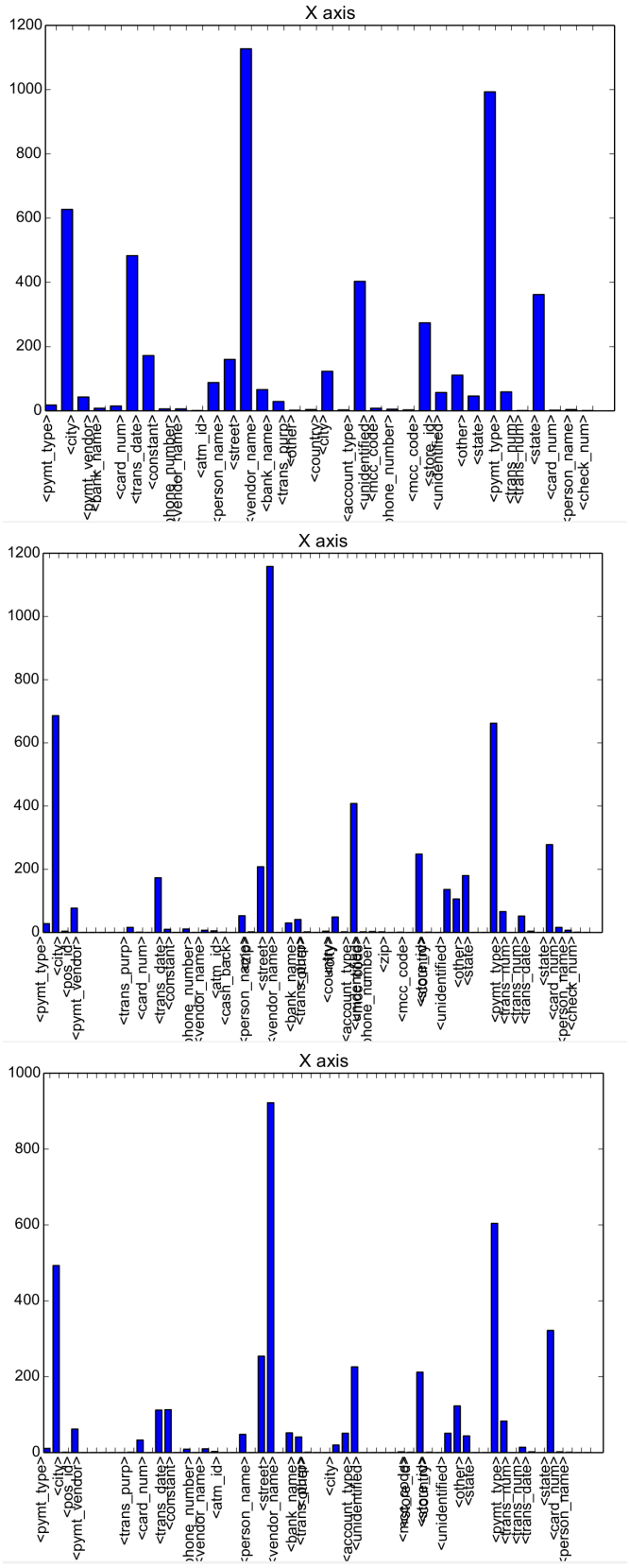

Figure 4.