Big Data Science Final Exam

Submitted by

- Rachit Jain
  - Net ID: rj2219
  - Email ID: rj2219@nyu.edu
- Rumi Desai
  - Net ID: rhd9863
  - Email ID: rhd9863@nyu.edu

Q1

As per the feedback received on the search engine, the searches are not contextual. We have divided our system design into various sub parts and will make them function as independent modules. To improve on the context of the search, we plan on updating the system as follows.

- Apache Solr stores each record as json type and uses an indexing mechanism to return the queries.
- Since the documents are of various kinds, we plan on including a **classifier for topic modeling** of the documents.
- The documents will be categorized into various topics such as setup related, process related, company wise segregation.
- The documents will then be sectioned into **different collections in Apache Solr** so that the indexing of those documents are returning relevant results.
- Another benefit of collections in Solr would be that since there are a lot of documents, the overall indexing would take a lot of time. With collections, this indexing time could be reduced by a lot (an exact figure would be presented when the implementation is complete).

- The **classifier** mentioned in the above points will be based on the **PyTorch framework** which is a deep learning framework. The classifier will be able to generate hidden patterns in the data for the topic modeling of the documents. The model will be able to perform extremely well in our setting as we will have a lot of data and will result in more accurate results.
- We also plan on building a semantic search model in the Solr framework by a **Collaborative Filtering mechanism**. This model will be able to cluster the users so that the searches become more relevant.
- This model will be able to filter the user and the relevant searches based on what the other users are searching. It is a model which consistently learns and updates based on how the other user **searches are being ranked.**
- The above collaborative filtering model will use **embeddings of the documents** to model similar users in a cluster. Advantages of using this algorithm would be that the searches here gain context by how the other user is utilizing our search engine.
- Another important aspect of our pipeline would be the retraining part where we will develop an **architecture of ranking the search results based on user feedback**. This is important since we want a mechanism for improving the precision and recall of the system.
- Each new document would be modeled based on the topic by using a **clustering algorithm** which will be an unsupervised model.
- We also plan on giving **recommendations** to the user while searching based on their **past searches** and the past searches of their cluster mates.

Based on the above outline, we plan on hiring individuals with knowledge of deep learning systems. Particularly, our approach would require individuals with knowledge of Apache Solr, Natural Language Processing, and PyTorch. We also plan on hiring individuals who have Systems knowledge and can implement Data Lakes. Due to time constraints, we are planning on using Hadoop as storage for now but are also simultaneously reviewing Apache Spark as our storage. Using Spark would improve our storage structure as the Hadoop storage is deprecating.

Hope this high level outline gives you a rough idea of how we plan on improving the system.

Best Regards,

Rachit and Rumi

Q2

1.

Metrics	Hadoop	Spark
Functionality	It is an open source framework that makes use of MapReduce Algorithm.	Spark is a cluster computing technology that utilizes the MapReduce model to perform many computations.
Cost	Due to reliance on disk storage for processing, it runs at a lower cost.	Since it requires a lot of RAM to run in-memory, thereby increasing the cluster and leading to a higher cost.
Data Processing	With Hadoop MapReduce, a developer can perform batch and linear data processing.	It can process real-time and live unstructured data stream processing like facebook and twitter.
Processing Speed	It reads and writes from a disk thereby slowing down the processing speed	Since it has an in-memory, it reduces the read/write cycles thereby decreasing the processing speed.
Performance	Slower performance since MapReduce reads and writes from disk.	Faster performance since it uses random access memory (RAM).
Latency	High latency framework without interactive mode.	Low latency computing with interactive mode.
Security	More secure as it uses multiple authentication and access control methods	Uses shared secret and event logging and can be integrated with Hadoop to become more secure.
Scalability	It quickly scales to handle the demand through Hadoop Distributed File System (HDFS).	It relies on fault tolerant HDFS for large volumes of data.

2. Downsides of using Spark are as follows:
    a. It does not have its own file management system, therefore it is dependent on other platforms/frameworks like Hadoop for the same.
    b. While the incoming live stream of data is divided into batches where each batch is called Spark RDD which are further processed using map,reduce

and join etc. Therefore, it is not exactly real time processing but it is more like near real-time processing of live data only.

c. It makes use of manual optimization of the spark jobs.

d. It does not support record based window criteria. It only has time-based window criteria.

e. When we use Spark with Hadoop, it leads to the small file issue. This is because in HDFS deals with a small number of large files rather than a large number of small files. Spark can retain a number of small files on the network and uncompress them but this is possible only if the entire file is on a single core. The process of unzipping the files in order takes a long time, and the RDDs must be repartitioned to increase the processing efficiency.

3. In the Hadoop implementation of K-Means algorithm, we have a mapper and a reducer. A single MapReduce iteration corresponds one iteration in the classical implementation of K-Means. The algorithm works by choosing random initial centroids and then converging to centroids based on the minimum distance parameter between a point and the centroids. In each iteration, the mapper is calculating the distance between the point and finding the closest centroid. The reducer is calculating the new centroid by getting all the cluster points after the iteration is done and then emitting the new centroid. The algorithm runs until a threshold is reached or the previous centroid the same as the new centroids.

```
def calculateMapper(centroid, inputPoint):
  minDistance, centroidResult = calculateMinDistance(centroid, inputPoint)
  EMIT(minDistance, centroidResult)

def calculateReducer(centroids, clusterPoints):
  newCentroid = calculateNewCentroid(centroids, clusterPoints)
  EMIT(newCentroid)

def KMeans():
  centroids = getCentroids()
  while (prevCentroids - centroids > threshold) or (prevCentroids == centroids):
    point, closestCentroid = calculateMapper(centroids, inputPoint)
    newCentroid = calculateReducer(centroids, clusterPoints)
    prevCentroids = centroids
    centroids = newCentroid
```

Q3

KNN

K-Nearest Neighbors (KNN) is basically a supervised learning algorithm used for both regression and classification. This algorithm divides our data in different classes and for a given test data point, it calculates the distance of that point with all the training points of different classes. Post that, it selects the K number of points which are closest to the test data.

It can be used for both classification and regression. For classification, the KNN algorithm calculates the probability of the test data belonging to the classes of ‘K’ training data and the class that holds the highest probability will be selected. In the case of regression, the value is the mean of the ‘K’ selected training points.

Examples : Recommendation systems on Netflix/Youtube can be a very good example of KNN algorithms.

```
def KNN(data, query):
  # input: data, query
  # output: predicted class labels

  X = readData(data)
  K = selectKBasedOnHueristic()
  for data in X:
    dist = calculateDistance(data, query)
    priorityQueue.append(dist)
  elem = topKElements(priorityQueue)
  meanData = calculateMean(elem)
  return meanData
```

Advantages:

1. Intuitive and simple to comprehend and execute
2. Can learn non-linear decision boundaries when used for classification and regression. Can come up with a highly flexible decision boundary adjusting the value of K.
3. The value of K is the only single hyperparameter hence tuning of this hyperparameter becomes easy.

Disadvantages:

1. With an increase in the data, computation becomes slower

2. It is sensitive to the outliers present in the dataset.

SVM

The Support Vector Machine is a supervised algorithm that works on classification problems. The objective of this algorithm is to find a hyperplane in an N-dimensional space. Each data is projected as a coordinate in an N-dimensional space. There can be multiple hyperplanes but the objective is to find the most optimal one in a way that it can work with unseen samples as well. The most optimal one here is the one which has the most margin with the data points. Support vectors are data points that are closer to the hyperplane and have an influence on the hyperplane's position and orientation.

Advantages of SVM:

- In high-dimensional scenarios, it works well.
- It saves memory by using a subset of training points termed support vectors in the
  decision function.
- For the decision functions, several kernel functions can be given, as well as bespoke
  kernels.

Disadvantages

- When the data set contains more noise, such as target classes that overlap, it also
  performs poorly.
- When we have a large data collection, it does not perform well because the training
  time is longer.

```
def SVM(data, classLabels, alpha):
  # input: data, classLabels
  # output: support vectors
```

```
  X = readData(data)
  C = calculateCBasedOnHeuristic()
  while no changes in alpha;
    for (x1, y1), (x2, y2) in X, Y:
      alpha = updateAlphaOptimization()
  return supportVectors
```

Naive Bayes

It is a probability algorithm used for classification of entities. The basic algorithm is based on the probability theory defined by Bayes which is trying to find conditional probability. The concept here is to find the probability of y given that X has already happened. For the prediction part, we will then take the arg max over the list of probability scores assigned. For our midterm problem consisting of the life insurance discounts, instead of using the decision tree, we could have also used the Naive Bayes formulation.

An important part here is to assume that all features are independent. This is the Naive assumption we are taking.

Advantages:

1. Algorithm performs well and fast.
2. The algorithm is able to handle higher dimension data.
3. Since the events are considered as independent, this algorithm is easy to parallelize.
4. It is a very simple model to implement and understand.

Disadvantages:

1. This model is generally a very naive model.
2. It does not give high accuracies on bigger datasets.
3. The naive assumption that features are independent is generally not the case.

```
def calculateNaiveBayes(X, probDistributionFeature):
  # input: probability distribution
  # output: probability
  return prior*probDistributionFeature(X[0]) / probDistributionFeature(X[1]
```

```
def naivebayes(data, labels):
  # input: Training dataset, Training labels
  # output: predicted class labels

  X = readData(data)
  meanData = calculateMean(X)
  stdData = calculateStd(X, meanData)
  probDistributionFeature = calculateProbDist(X[:, 0], meanData, stdData)
  probDistributionLabel = calculateProbDist(X[:, 1], meanData, stdData)
  predictedProbLabel1 = calculateNaiveBayes(X, probDistributionFeature)
  predictedProbLabel2 = calculateNaiveBayes(X, probDistributionLabel)
  return predictedProbLabel1*100, predictedProbLabel2*100
```

Random Forest

It is a supervised machine learning algorithm that comprises various decision trees to reach a single decision. It basically is an extension of the bagging method since it utilizes both bagging and feature randomness to create an uncorrelated forest of many decision trees.

Example: It can be used in the banking sector to predict the creditworthiness of an applicant applying for a loan.

Advantages:

1. Reduces the risk of overfitting which is generally present if we use the decision tree alone.
2. Useful for both the classification and regression tasks which higher accuracy since it is an ensemble method
3. It has various metrics to understand the feature importance required for the task and hence makes it easier

Disadvantages:

1. Very complex since it makes use of a collection of random forests.
2. For the same reason, it is more time consuming as well.

```
def randomForest(data, features):
  # input: data, features
  # output: predicted class labels

  X = readData(data)
  K = selectKBasedOnHueristic()
  N = selectNBasedOnHueristic()
  while N > 0:
```

```
    while calculateTotalNodes() == desiredNodes:
      topFeatures = selectTopKFeatures(features, K)
      bestSplitNode = calculateBestSplitPoint(data, topFeatures)
      childSplitNodes = calculateChildSplitPoint(data, bestSplitPoint)
    forest.append(childSplitNodes)
    N -= 1
  return forest
```

Apriori

1. The algorithm can be seen in the photos below:

Dataset :

Transaction ID. → List of purchased items.

T1 → I1, I2
T2 → I1, I2, I5
T3 → I1, I3
T4 → I1, I2, I3, I4
T5 → I2, I3

→ Applying the various steps of Apriori Algorithm on our dataset.

1) Listing the count of each item

Item	Count
I1	4
I2	3
I3	3
I4	1
I5	1

2) Pruning Step. : Deleting entries with count < 2 (∵ min. support = 2)

Item	Count
I1	4
I2	3
I3	3

3) Joining the items to forming the itemset

Item	Count
I1, I2	3
I1, I3	2
I2, I3	2

4) Pruning Step – same condition as before

Item	Count
I1, I2	3
I1, I3	2
I2, I3	2

5) Joining and Pruning Step. – Forming 3 itemset with the same condition as before.

Item	Count
I1, I2, I3	1

Now, the algorithm terminates since the set is empty.

∴ The frequent itemsets are {I1, I2}, {I1, I3}, {I2, I3}

Applying the association rules

{I1} ⇒ {I2}

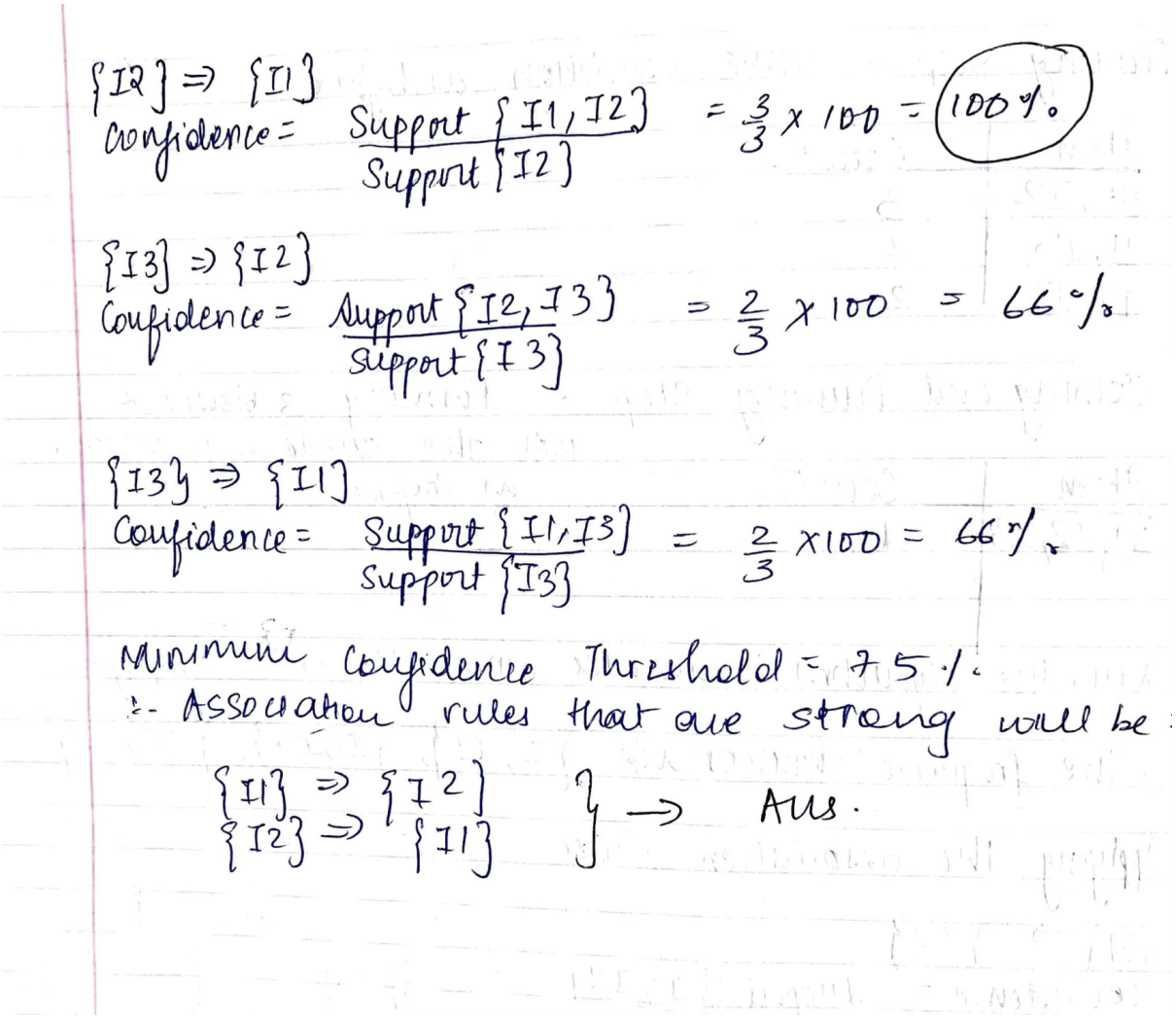
Confidence = $\frac{\text{support } \{I1, I2\}}{\text{support } \{I1\}} = \frac{3}{4} \times 100 =$ 75%.

{I1} ⇒ {I3}

Confidence = $\frac{\text{Support } \{I1, I3\}}{\text{Support } \{I1\}} = \frac{2}{4} \times 100 =$ 50%

{I2} ⇒ {I3}

Confidence = $\frac{\text{Support } \{I2, I3\}}{\text{Support } \{I2\}} = \frac{2}{3} \times 100 =$ 66%



2. The Apriori algorithm can be improved in two ways:
    a. By using a hashing technique, we can use a hash table to generate k-itemsets and the count of each by using a hash function.
    b. By using a transaction reduction, we can improve the efficiency. If a transaction does not have some frequent k-itemset, then it cannot include (k+1)-itemset. Therefore that transaction can be marked or deleted from further transactions.
3.

We can use the Decaying Window Algorithm to identify the most frequent elements that have occurred when we have an input streaming data.

Since it is streaming data, the elements in data are now basically a stream of baskets and not individual itemsets.

In this algorithm, we basically assign a score/weight to every element of the incoming data stream. Once this is done, we calculate the aggregate sum for each distinct element by simply adding all the weights assigned to that particular element. Now, the element with the highest total score is listed as the most frequent/popular/trending element.

It works in such a way that we assign higher weight to the newer elements in the stream.

We can do that by reducing the weight of the existing elements by a factor k(constant) and then assign this new element with a specific weight.

So basically whenever a new element arrives :

1. Multiply all previous counts by 1 − C. (C is usually a small constant of the order 10-6)
2. Add a new itemset with an initial count of 1.
3. Add 1 to an existing itemsets count.

Advantage :

We can not only use it to measure the most frequent elements in an input data stream but we can also basically keep a note of any sudden spikes in our data.

Example :

Streaming data consists of the following twitter tags:

Apple, Microsoft, Apple, Microsoft , Microsoft, Microsoft, Apple

Assumptions for simpler calculations -

Every element in this data has a weight of 1

c= 0.1

Calculating the aggregated sum for every tag

Apple:

Apple = 1 * (1-0.1) = 0.9

Microsoft = 0.9* (1-0.1) + 0 = 0.81 (Added 0 since current tag is different from Apple)

Apple = 0.81* (1-0.1) + 1 = 1.729 (Added 0 since same as Apple)

Microsoft = 0.9* (1-0.1) + 0 = 1.5561

Microsoft = 0.9* (1-0.1) + 0 = 1.4005

Microsoft = 0.9* (1-0.1) + 0 = 1.2605

Apple = 0.81* (1-0.1) + 1 = 2.135

Microsoft:

Apple = 0 * (1-0.1) = 0

Microsoft = 0* (1-0.1) + 1 = 1

Apple = 1* (1-0.1) + 0 = 0.9 (Added 0 since same as Apple)

Microsoft = 0.9* (1-0.1) + 1 = 1.81

Microsoft = 0.9* (1-0.1) + 1 = 2.7919

Microsoft = 0.9* (1-0.1) + 1 = 3.764

Apple = 0.81* (1-0.1) + 0 = 3.7264

Here, the score for Apple = 2.135 but for Microsoft= 3.7264

Therefore, Microsoft is Most Frequent in this series.

4. **Effective Sampling for Mining Association Rules - Yanroung Li et al**

In a large database, identifying the association rules becomes complex and time consuming. Using Sampling we can reduce the cost of mining significantly since mining algorithms now will need to deal with a smaller database instead of the original database.

For our sampling to be effective, it is very important to understand how this sampling of our data happens - determine the size of the sample for a given confidence level and error bound for the data mining process.

In this paper, they have improved the performance by employing the following steps :

1. From the original transaction Database D consisting of n transactions, a sample of size N is chosen.
2. The Sampling method used- Random Sampling Database with Replacement. Here, the sampling with replacement obtains n units independently and at each step every unit in the population has an equal probability of being included in the sample. Bernoulli's Binomial Distribution is the process chosen to select the sample. Total number of trials are n. But there are only two outcomes for each trail (both being complementary) i.e an item would either appear or not in the transaction. Probability p if it remains same from trial to trial and q being the probability that the transaction does not appear in that trial.

3. Determining the sufficient sample size using the Central Limit Theorem for a given error bound and confidence level which is much smaller than that based on the Chernoff Bounds [2,3]
4. The errors we achieved are within the given error bounds and the accuracy has increased

The conclusion derived at the end of this process was that sampling provides good estimation of complete frequent itemsets where approximate results are acceptable.

Decision Trees

Q5] Decision Trees:

Dataset:

Weekend	Weather	Parents	Money	Decision
W1	Sunny	Yes	Rich	Cinema
W2	Sunny	No	Rich	Tennis
~~W3~~	~~Sunny~~			
W4	Rainy	Yes	Rich	cinema
W5	Rainy	No	Rich	Stay In
W6	Rainy	Yes	Poor	cinema

1) Finding the root node by calculating Entropies:

For simplicity, let us divide "Decision" into { Go out, Stay In }

Weekend	Weather	Parents	Money	Decision
W1	Sunny	Yes	Rich	Go out
W2	Sunny	No	Rich	Go out
W4	Rainy	Yes	Rich	Go out
W5	Rainy	No	Rich	Stay in
W6	Rainy	Yes	Poor	Go out

∴ Stay In → 1
Go out → 4.

$$\text{Entropy}(S) = \sum_{x \in X} -P(x) \log_2 P(x)$$

$$= -\frac{1}{5} \log_2\left(\frac{1}{5}\right) - \frac{4}{5} \log_2\left(\frac{4}{5}\right)$$

$$= 0.7219$$

Conditional Entropy:

(i) Weather:

$$H(S_{sunny}) = 0$$
$$H(S_{rainy}) = -\frac{1}{3}\log_2\left(\frac{1}{3}\right) - \frac{2}{3}\log_2\left(\frac{2}{3}\right)$$
$$= 0.9182$$

Conditional Entropy $= P(S_{sunny}) \cdot H(S_{sunny}) + P(S_{rainy}) \cdot H(S_{rainy})$
$$= \frac{2}{5}(0) + \frac{3}{5}(0.9182)$$
$$= 0.5509$$

$IG(S, Weather) = 0.7219 - 0.5509$
$= 0.1709$

(ii) Parents:

$H(S_{yes}) = 0$
$H(S_{no}) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1$

Conditional Entropy $= P(S_{yes}) \cdot H(S_{yes}) + P(S_{no}) \cdot H(S_{no})$
$$= \frac{3}{5}(0) + \frac{2}{5}(1)$$
$$= 0.4$$

$IG(S, Weather) = H(S) - H(S|Weather)$
$= 0.3219$

(ii) Money:

$H(S_{poor}) = 0$

$H(S_{rich}) = -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right)$

$= 0.8112$

$H(S|Weather) = \frac{1}{2}(0) + \frac{4}{5}(0.8112) = 0.6489$

$IG(S, Weather) = 0.7219 - 0.6489 = 0.073$

Parents has highest IG ∴ Root Node.

2) Same procas for other nodes

Weather	Money	Decision
Sunny	Rich	Go out
Rainy	Rich	Stay in

Entropy (S) $= -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right)$

$= 1$

(i) Weather:

$H(S_{sunny}) = 0$

$H(S_{rainy}) = 0$

Conditional Entropy $= \frac{1}{2}(0) + \frac{1}{2}(0) = 0$.

$IG(S, Weather) = 1 - 0 = 1$

(ii) Money :

$H(S_{poor}) = 0$

$H(S_{rich}) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\frac{1}{2}$

$= 1$

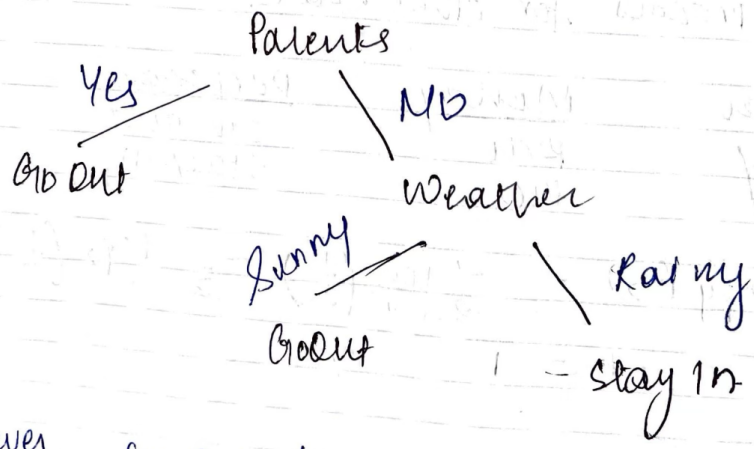
Conditional Entropy $= \frac{2}{2}(0) + \frac{2}{2}(1)$

$= 1$

$IG(S, \text{Weather}) = 1 - 1 = 0$

∴ Weather → highest IG

∴ Decision Tree :



However, Go Out / Stay In was our choice.

Tennis → 1 Cinema → 3

∴ Entropy = $-\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right)$

$= 0.8112$

(i) Weather :

$H(S_{sunny}) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right)$

$= 1$

$H(S_{rainy}) = 0$

C.E $= \frac{2}{4}(1) + \frac{2}{4}(0) = 0.5$

IG(S, Weather) $= 0.8112 - 0.5 = 0.3112$

(ii) Parents:

$H(S_{yes}) = 0$

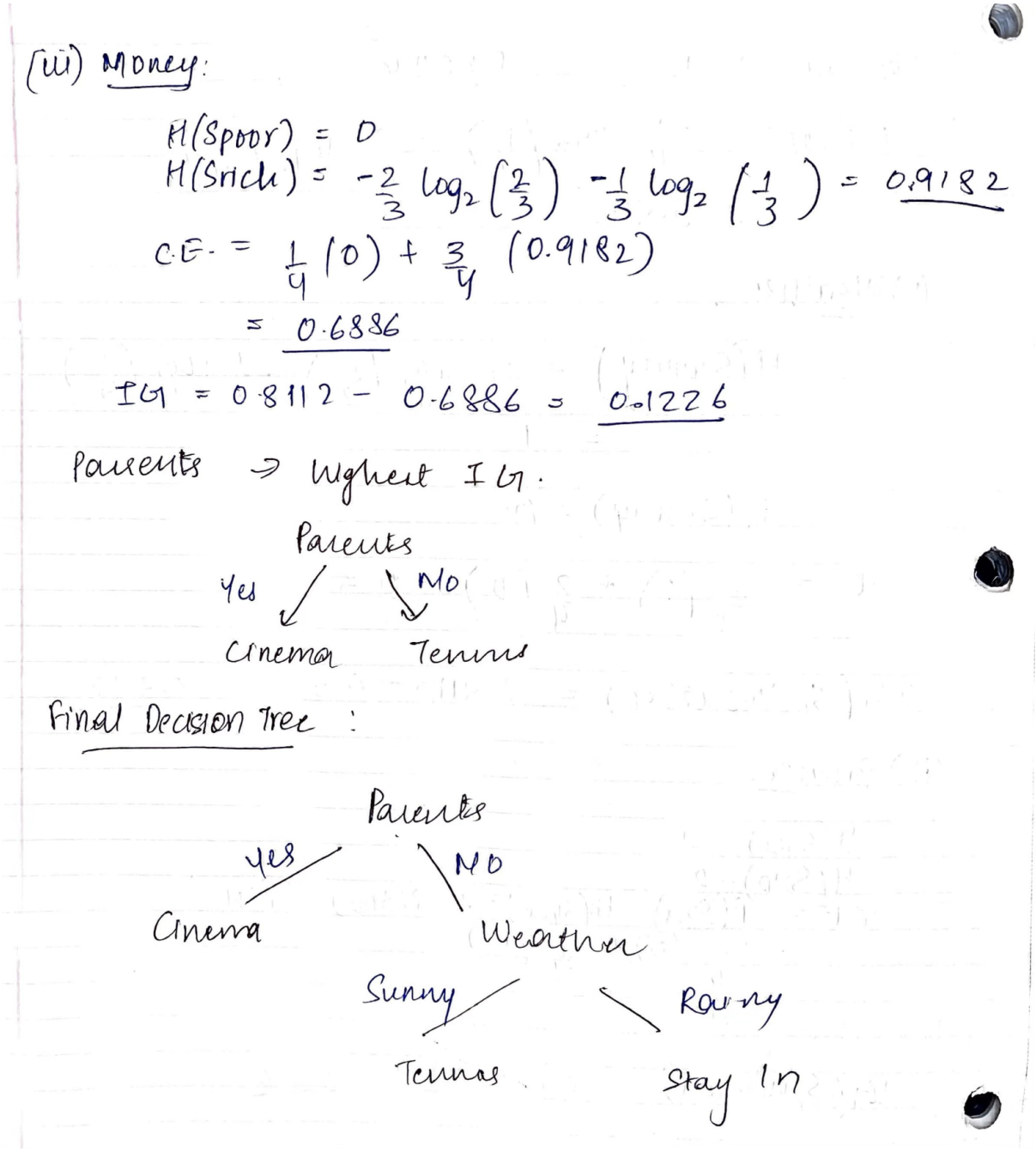
$H(S_{NO}) = 0$

C.E $= P(S_{yes}) \cdot H(S_{yes}) + P(S_{no}) \cdot H(S_{No})$

$= \frac{3}{4}(0) + \frac{1}{4}(0)$

$= 0$

IG(S/Weather) $= 0.8112 - 0 = 0.8112$



Customer Segmentation and Churn

A. Customer Segmentation is the process of segmenting customers into various categories. These categories include segmentation based off of age, gender, location, country, etc. It is important as different customers can churn due to different reasons. So it is important to create strategies to avoid this churning. The strategies can differ based on how the customer is segmented. By segmenting, there can be

same strategies which can be used for a specific set of segmented customers thereby reducing effort in devising different strategies for each segment.

B. Customer churn is when a customer stops using the company's product or services. Customer churn rate is the rate at which the number of customers stop doing business with a company over a specified period of time. It is important to predict the customer churn rate as it is important for the company to know if they are losing customers. It is important since a company should know if there are some policies or marketing techniques which are resulting in customers leaving their business. Customer churn is calculated by calculating the customer churn rate.

C. For prediction of customer churn, first we will collect data. In the collected data, we will assign the label roles to the class label and then select features which is choosing what features are important. The next step is to transform features and sample the data. Sampling the data here means making the classes to be of similar sizes because for our specific example, the number of people who churn are not as high as the number of people who do not churn. After this, it helps to divide the features into categories of Recency, Frequency, and Monetary to help understand the model better. Once this step is complete, we will reduce the dimensionality of the data so as to make better visualizations and remove unnecessary features. Once we are done with the all the preprocessing steps, we will model the data based on a few algorithms. Based on our findings, we report that Random Forest model performed the best and gave good results. The recall is the most important metric here since accuracy can be high due to sampling techniques. But it is important for the model to understand what features to select and how much weight should be assigned so as to predict churn. In our modeling, we applied Logistic Regression, Random Forest, and a Neural Network. We run a cross validation set 10 folds so that the model does not overfit.

It is important that we choose good preprocessing techniques such as PCA, sampling, feature transformation for accurate results. Once the preprocessed data is good, there can be multiple modeling approaches that can be applied to achieve good results.

References

[1] https://www.geeksforgeeks.org/difference-between-hadoop-and-spark/

[2] https://www.ibm.com/cloud/blog/hadoop-vs-spark

[3] https://data-flair.training/blogs/limitations-of-apache-spark/

[4] https://www.knowledgehut.com/blog/big-data/apache-spark-advantages-disadvantages

[5] https://github.com/seraogianluca/k-means-mapreduce

[6] https://machinelearningmastery.com/classification-as-conditional-probability-and-the-naive-bayes-algorithm/

[7] https://www.ibm.com/cloud/learn/random-forest

[8] https://www.softwaretestinghelp.com/apriori-algorithm/

[9] https://nitinbhojwani-tech-talk.blogspot.com/2018/12/decaying-window-algorithm.html

[10] https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.9385&rep=rep1&type=pdf

[11] https://opensourceconnections.com/blog/2013/08/25/semantic-search-with-solr-and-python-numpy/

[12] https://link.springer.com/content/pdf/10.1007/978-3-540-30549-1_35.pdf