Big Data Science: Final Exam

Due on May 17, 2022 at 11:59pm

Professor Anasse Bari

Maya Balaji (N17210260)
Deeksha Tiwari (N12468048)

Problem 1

Hi Team,

We have investigated the issue of search results turning up irrelevant to the user's query and found that this is due to an absence of **contextual search** in the search engine. To improve the context of the searches, we propose the following changes to the system design:

- Performing **intent classification** of the documents which are of varied types which will help to ingest and organize the data. This can be done using an NLP engine having the ability to integrate with multiple content repositories and databases, handle different file formats, and categorize the documents. The text classifier will be based on the PyTorch deep learning framework. Since Apache Solr stores the data in json format and returns queries through an indexing mechanism, the documents can be partitioned into various collections to make the indexing easier and make the returned results more relevant. Examples of categories: Process, Company, Setup, etc.
- Using a **Collaborative Filtering** mechanism to build a semantic search model using the Solr framework that will enable clustering similar users which will further improve the search results. This model will consistently learn and update itself based on the ranking of other user searches. This will also contribute to the currently missing context in the search engine. The clustering algorithm will be an unsupervised model.
- Finally, the **retraining** module needs to be optimized so as to rank the search results based on user feedback. This is necessary to improve the precision and recall of the system.

Each feature will be implemented as an independent module and to do so effectively we are working on hiring talent with a comprehensive understanding of deep learning systems. The required skillset would be Apache Solr, PyTorch, and other NLP frameworks and techniques. Currently Hadoop is being used for storage, however, we can consider a shift to Apache Spark in the future to further increase performance. Another potential addition to the system design can be to provide recommendations to the user based on their own search history as well as those of other users in the same cluster.

The outline provided above should give the team a high level idea on the plan of action for improving the system. Please reach out if there are any queries or concerns.

Regards,
Maya and Deeksha

Problem 2

(1) **Differences between Apache Hadoop and Spark:**

-	**Hadoop**	**Spark**
Cost	Due to reliance on disk storage for processing, it runs at a lower cost.	Due to reliance on in-memory real-time data processing, it runs at a higher cost.
Processing	Ideal for batch and linear data processing.	Ideal for real-time and live unstructured data stream processing.
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 Spark:**

- **Small File issue:** This is a major issue faced when using Spark with Hadoop because in HDFS we are provided with a small number of large files rather than a large number of small files. Spark works to keep 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. This costs a significant amount of time in unzipping the files in sequence, moreover the RDDs need to be repartitioned to improve the processing efficiency.
- **Manual Optimization:** To partition and cache in Spark correctly, it needs to be controlled and optimized manually. Moreover it is adequate only to specific datasets. This is done by passing a number of partitions as the second parameter to the parallelize method.

(3) **K-means in Map-Reduce:**

Pseudocode:

Input: Given point and set of centroids
Output: New centroid
Data: centroids are k random sampled points from the dataset

```
/* Map function computes the distance between the data point and each centroid and
   emits index of closest centroid and the point                                  */
Function Mapper(point):
    min_distance = INFINITY
    closest_centroid = -1
    for centroid in centroid_list do
        distance = distance(centroid, point)
        if distance < min_distance then
            closest_centroid = index_of(centroid)
            min_distance = distance
        end
    end
    emit(closest_centroid, point)

/* Combine function sums all data points belonging to a cluster and computes the
   centroid as the arithmetic mean position of the points and emits index of
   centroid amd sum of points.  This helps reduce amount of data transmitted to
   reducer.                                                                       */
Function Combiner(centroid_index, points_list):
    point_sum.number_of_points = 0
    point_sum = 0
    for point in points_list do
        point_sum += point
        point_sum.number_of_points += 1
    end
    emit(centroid_index, point_sum)

/* Reduce function collects all points belonging to a cluster and computes the new
   centroid as the arithmetic mean position of the points and emits new centroid  */
Function Reducer(centroid_index, point_sum_list):
    number_of_points = partial_sum.number_of_points
    point_sum = 0
    for partial_sum in partial_sum_list do
        point_sum += partial_sum
        point_sum.number_of_points += partial_sum.number_of_points
    end
    centroid_value = point_sum / point_sum.number_of_points
    emit(centroid_index, centroid_value)
```

Problem 3

Part A: Algorithms

(1) KNN: The KNN algorithm assumes that objects that are similar are nearby. To put it another way, similar things are close together. KNN is a mathematical representation of similarity (also known as distance, proximity, or closeness). KNN works by calculating the distances between a query and all of the instances in the data, picking the K closest examples to the query, and then voting for the most frequent label in the case of classification or averaging the labels in the case of regression. Recommender systems are a good example of KNN algorithms. Therefore, at scale, Amazon items, Medium articles, Netflix movies, or YouTube videos.

Advantages of KNN:

- The algorithm is simple and easy to implement.
- There's no need to create a model, tweak several parameters, or make any more assumptions.
- The algorithm is extremely adaptable. It has classification, regression, and search capabilities.

Disadvantages of KNN:

- As the number of samples and/or predictors/independent variables grows, the method becomes much slower.

Pseudocode:

Input: Training set D, test object x, category label set C
Output: Category c_x of x which belongs to C
Function `KNN(`*D, x, C*`)`**:**
  for *y in D* **do**
    distance = D(y, x)
  end
  compute set I containing indices of k smallest distances
  c_x = majority label from I
  return c_x

(2) SVM: The Support Vector Machine or SVM is a supervised machine learning algorithm that can solve classification and regression problems. Each data item is plotted as a point in n-dimensional space where n is the number of features, with the value of each feature being the value of a certain coordinate in the SVM algorithm. Then we accomplish classification by locating the hyper-plane that clearly distinguishes the two classes. Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line). The SVM classifier is a frontier that separates the two classes (hyper-plane and line) the best. Hyperplanes are decision boundaries that

aid in data classification. Different classes can be assigned to data points on either side of the hyperplane. There are numerous hyperplanes from which one can choose to split the two groups of data points. The goal is to discover a plane with the greatest margin, or the largest distance between data points from both classes. Maximizing the margin distance gives some reinforcement, making it easier to classify future data points. Support vectors are data points that are closer to the hyperplane and have an influence on the hyperplane's position and orientation. We maximize the classifier's margin by using these support vectors. The hyperplane's position will vary if the support vectors are deleted.

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 of SVM:

- When we have a large data collection, it does not perform well because the training time is longer.
- When the data set contains more noise, such as target classes that overlap, it also performs poorly.
- Probability estimates are produced using a time-consuming five-fold cross-validation method, which is not directly provided by SVM. It's part of Python's scikit-learn library's related SVC algorithm.

Pseudocode:

Input: X and Y with training labeled data, α
Output: Support Vectors
Function `SVM(`X, Y, α`)`**:**
  $C \leftarrow 10$ (some value)
  repeat
    for $\{x_i, y_i\}$, $\{x_j, y_j\}$ *in* X *and* Y **do**
      Optimize α_i and α_j
    end
  until *no changes in* α *or other resource constraint criteria met*

(3) Naive Bayes: Naive Bayes is a probabilistic algorithm typically used in classification problems. The basic concept is to find conditional probability of Y given that X has already happened. For the purpose of prediction, we take the maximum from the the list of assigned probability scores. An important assumption is that all features are independent (Naive assumption). Example of Naive Bayes is spam filters on email applications.

Advantages of Naive Bayes:

- It is a very simple model in terms of understanding and implementation.
- It is easy to parallelize since the events are considered independent.
- It performs well and fast.
- It can handle higher dimensionionality of data.

Disadvantages of Naive Bayes:

- The naive assumption that features are independent is not the case realistically
- The accuracy is low for bigger datasets.

Pseudocode:

Input: Training Set T, Predictor Variable $F = (f_1, f_2, \ldots, f_n)$
Output: Class of testing dataset
Function `NaiveBayes(`T, F`)`**:**
    Read training dataset T
    Calculate mean and standard deviation of predictor variables in each class
    repeat
        for f_i *in* F **do**
            Calculate probability of f_i using gauss density equation in each class
        end
    until *the probability of all predictor variables has been calculated*
Calculate the likelihood for each class
max = greatest likelihood
return max

(4) Random Forests: It is a commonly used supervised machine learning algorithm which combines the output of various decision trees to reach a single output. It works as an extension of the bagging method since it creates an uncorrelated forest of decision tree by utilizing both bagging and feature randomness. Example of Random Forest is identifying stock behavior (profit/loss) in the stock market.

Advantages of Random Forest:

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

Disadvantages of Random Forest:

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

Pseudocode:

Input: Training Set T, Features F, Number of trees in forest K
Output: Decision Tree
Function `RandomForest(`T, F, K`):`

```
P ← φ
for i in K do
    T_i ← a bootstrap sample from T
    p_i ← RandomizedTreeLearn(T_i, F)
    P ← P ∪ {p_i}
end
return P
```

Function `RandomizedTreeLearn(`T, F`):`

```
At each node:
    f ← very small subset of F
    Split on best feature in f
return The learned tree
```

Part B: Apriori
(1) Let us apply Apriori Algorithm to the given dataset:

Step 1: Count of each item

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

Step 2: Prune Step - we delete the entries with count $<$ min_support ($= 2$)

Item	Count
I1	4
I2	3
I3	3

Step 3: Join Step - Form 2-itemset

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

Step 4: Prune Step - we delete the entries with count < min_support

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

Step 5: Join and Prune Step - Form 3-itemset and delete the entries with count < min_support

Item	Count
I1, I2, I3	1

The algorithm terminates here since the set is empty.

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

Now we can generate association rules:
{I1} $\implies$ {I2} Confidence = Support{I1, I2} / Support{I1} = (3/4)*100 = **75%**
{I1} $\implies$ {I3} Confidence = Support{I1, I3} / Support{I1} = (2/4)*100 = 50%
{I2} $\implies$ {I3} Confidence = Support{I2, I3} / Support{I2} = (2/3)*100 = 66%
{I2} $\implies$ {I1} Confidence = Support{I1, I2} / Support{I2} = (3/3)*100 = **100%**
{I3} $\implies$ {I2} Confidence = Support{I2, I3} / Support{I3} = (2/3)*100 = 66%
{I3} $\implies$ {I1} Confidence = Support{I1, I3} / Support{I3} = (2/3)*100 = 66%

We can see that only two association rules are above the given minimum confidence threshold of 75%, hence the strong association rules are:

{I1} $\implies$ {I2}
{I2} $\implies$ {I1}

(2) Two algorithmic methods that can be used to improve the efficiency of the Apriori's algorithm:

- **Hashing technique:** In this a hash table is used to generate k-itemsets and the count of each by use of a hash function.
- **Partitioning:** Through this method, only two database scans are needed to mine the frequent itemsets since it must be frequent in at least one of the partitions of the database to be considered a potential frequent itemset for the output.

(3) The **Decaying Window Algorithm** can be used to identify the frequent itemset for an input of streaming data. Since it is streaming data, the elements are a stream of baskets and not individual itemsets. In this algorithm, a weight is assigned to every element of the

incoming data stream and then the aggregate sum for each distinct element is calculated by simply adding all the weights assigned to that particular element. Finally, the element with the highest total weight is returned as the most frequent element. The newer elements in the stream get assigned higher weights which can be done by reducing the weight of the existing elements by a constant factor of k and then assigning the new element with a specific weight.

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 itemset's count.

Advantage: We can use it not only to measure the most frequent elements from an input data stream but we can also track any sudden spikes in the data.

Example : Streaming data consists of the following tags - Xbox, PS5, Xbox, PS5 , PS5, PS5, Xbox

Assumptions for simpler calculations -

- Every element in this data has a weight of 1
- $C = 0.1$

Let us calculate the aggregated sum for every tag -
Xbox:
Xbox $= 1 * (1 - 0.1) = 0.9$
PS5 $= 0.9 * (1 - 0.1) + 0 = 0.81$ (Adding 0 since current tag is different)
Xbox $= 0.81 * (1 - 0.1) + 1 = 1.729$ (Adding 1 since tag is same)
PS5 $= 0.9 * (1 - 0.1) + 0 = 1.5561$
PS5 $= 0.9 * (1 - 0.1) + 0 = 1.4005$
PS5 $= 0.9 * (1 - 0.1) + 0 = 1.2605$
Xbox $= 0.81 * (1 - 0.1) + 1 = 2.135$

PS5:
Xbox $= 0 * (1 - 0.1) = 0$
PS5 $= 0 * (1 - 0.1) + 1 = 1$
Xbox $= 1 * (1 - 0.1) + 0 = 0.9$ (Adding 0 since tag is same)
PS5 $= 0.9 * (1 - 0.1) + 1 = 1.81$
PS5 $= 0.9 * (1 - 0.1) + 1 = 2.7919$
PS5 $= 0.9 * (1 - 0.1) + 1 = 3.764$
Xbox $= 0.81 * (1 - 0.1) + 0 = 3.7264$

The score for PS5 is higher, therefore, PS5 is the Most Frequent Element.

(4) An approach which speeds up association rule mining using Sampling:

Input: A relation r over a binary schema R, a frequency threshold min_f_r, subset S of $F(r, min\_f_r)$

Output: The collection $F(r, min\_f_r)$ of frequent sets and their frequencies

Function `FrequentSetDiscovery(`T, F`):`

  repeat

    compute $S := S \cup B_{d^-}(S)$

    `// using concept of negative border`

  until S *does not grow*

compute $F := X \in S \mid f_r(X, r) \geq min\_f_r$

for x *in* $X \in F$ **do**

  output X and $f_r(X, r)$

end

(5) Let us apply the ID3 decision tree algorithm to derive the decision tree from the given dataset -

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

Step 1: Create a Root Node through calculation of entropy:
Let us divide the decision feature into 2 values "Stay In" and "Go out" to keep it binary

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

We get the counts as follows:

Stay In	**Go out**
1	4

$$\begin{aligned} Entropy(S) &= \sum_{x \in X} -P(x) \log_2 P(x) \\ &= -\left(\frac{1}{5}\right) \log_2 \left(\frac{1}{5}\right) - \left(\frac{4}{5}\right) \log_2 \left(\frac{4}{5}\right) \\ &= 0.7219 \end{aligned}$$

Now we calculate the Conditional Entropy of other attributes to get the information gain -

(i) Weather:

$$
\begin{aligned}
H(S_{sunny}) &= 0 \ (No\ Variation) \\
H(S_{rainy}) &= -\left(\frac{1}{3}\right)\log_2\left(\frac{1}{3}\right) - \left(\frac{2}{3}\right)\log_2\left(\frac{2}{3}\right) \\
&= 0.9182 \\
Conditional\ Entropy &= \sum_{x \in X} P(x) \cdot H(x) \\
&= P(S_{sunny}) \cdot H(S_{sunny}) + P(S_{rainy}) \cdot H(S_{rainy}) \\
&= \left(\frac{2}{5}\right) \cdot 0 + \left(\frac{3}{5}\right) \cdot 0.9182 \\
&= 0.5509 \\
Information\ Gain,\ IG(S, Weather) &= H(S) - H(S|Weather) \\
&= 0.7219 - 0.5509 \\
&= 0.1709
\end{aligned}
$$

(ii) Parents:

$$
\begin{aligned}
H(S_{Yes}) &= 0 \ (No\ Variation) \\
H(S_{No}) &= -\left(\frac{1}{2}\right)\log_2\left(\frac{1}{2}\right) - \left(\frac{1}{2}\right)\log_2\left(\frac{1}{2}\right) \\
&= 1 \\
Conditional\ Entropy &= \sum_{x \in X} P(x) \cdot H(x) \\
&= P(S_{Yes}) \cdot H(S_{Yes}) + P(S_{No}) \cdot H(S_{No}) \\
&= \left(\frac{3}{5}\right) \cdot 0 + \left(\frac{2}{5}\right) \cdot 1 \\
&= 0.4 \\
Information\ Gain,\ IG(S, Weather) &= H(S) - H(S|Weather) \\
&= 0.7219 - 0.4 \\
&= \mathbf{0.3219}
\end{aligned}
$$

(iii) Money:

$$
\begin{aligned}
H(S_{poor}) &= 0 \ (No\ Variation) \\
H(S_{rich}) &= -\left(\frac{3}{4}\right)\log_2\left(\frac{3}{4}\right) - \left(\frac{1}{4}\right)\log_2\left(\frac{1}{4}\right) \\
&= 0.8112 \\
Conditional\ Entropy &= \sum_{x \in X} P(x) \cdot H(x) \\
&= P(S_{poor}) \cdot H(S_{poor}) + P(S_{rich}) \cdot H(S_{rich}) \\
&= \left(\frac{1}{5}\right) \cdot 0 + \left(\frac{4}{5}\right) \cdot 0.8112 \\
&= 0.6489 \\
Information\ Gain,\ IG(S, Weather) &= H(S) - H(S|Weather) \\
&= 0.7219 - 0.6489 \\
&= 0.073
\end{aligned}
$$

The attribute **Parents** has the highest Information Gain as seen above, so it is chosen as the root node.
Step 2: Find entropy of all observations and again select attribute with highest information gain until all are covered:
With Parents as root node, we see that there is no variation in outcome for "Yes", hence it will result in a leaf node of "Go out"
Now we can check attributes for the "No" branch -

Weather	**Money**	**Decision**
Sunny	Rich	Go out
Rainy	Rich	Stay In

$$
\begin{aligned}
Entropy(S) &= \sum_{x \in X} -P(x)\log_2 P(x) \\
&= -\left(\frac{1}{2}\right)\log_2\left(\frac{1}{2}\right) - \left(\frac{1}{2}\right)\log_2\left(\frac{1}{2}\right) \\
&= 1
\end{aligned}
$$

We calculate the Conditional Entropy of other attributes to get the information gain -

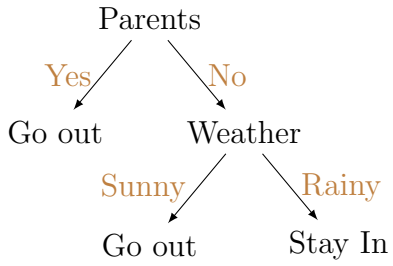
(i) Weather:

$$
\begin{aligned}
H(S_{sunny}) &= 0 \ (No\ Variation) \\
H(S_{rainy}) &= 0 \ (No\ Variation) \\
Conditional\ Entropy &= \sum_{x \in X} P(x) \cdot H(x) \\
&= P(S_{sunny}) \cdot H(S_{sunny}) + P(S_{rainy}) \cdot H(S_{rainy}) \\
&= \left(\frac{1}{2}\right) \cdot 0 + \left(\frac{1}{2}\right) \cdot 0 \\
&= 0 \\
Information\ Gain,\ IG(S, Weather) &= H(S) - H(S|Weather) \\
&= 1 - 0 \\
&= 1
\end{aligned}
$$

(ii) Money:

$$
\begin{aligned}
H(S_{poor}) &= 0 \ (No\ Variation) \\
H(S_{rich}) &= -\left(\frac{1}{2}\right) \log_2 \left(\frac{1}{2}\right) - \left(\frac{1}{2}\right) \log_2 \left(\frac{1}{2}\right) \\
&= 1 \\
Conditional\ Entropy &= \sum_{x \in X} P(x) \cdot H(x) \\
&= P(S_{poor}) \cdot H(S_{poor}) + P(S_{rich}) \cdot H(S_{rich}) \\
&= \left(\frac{0}{2}\right) \cdot 0 + \left(\frac{2}{2}\right) \cdot 1 \\
&= 1 \\
Information\ Gain,\ IG(S, Weather) &= H(S) - H(S|Weather) \\
&= 1 - 1 \\
&= 0
\end{aligned}
$$

The attribute **Weather** has the highest Information Gain as seen above, so it is chosen as the next node.
Now our decision tree is -



Since the outcome "Go out" has 2 possible values (Cinema, Tennis) according to our initial table, we can apply the above steps again to get the decision tree for that -

Weekend	**Weather**	**Parents**	**Money**	**Decision**
W1	Sunny	Yes	Rich	Cinema
W2	Sunny	No	Rich	Tennis
W4	Rainy	Yes	Rich	Cinema
W6	Rainy	Yes	Poor	Cinema

We get the counts as follows:

Tennis	**Cinema**
1	3

$$\begin{aligned} Entropy(S) &= \sum_{x \in X} -P(x) \log_2 P(x) \\ &= -\left(\frac{1}{4}\right) \log_2 \left(\frac{1}{4}\right) - \left(\frac{3}{4}\right) \log_2 \left(\frac{3}{4}\right) \\ &= 0.8112 \end{aligned}$$

Now we calculate the Conditional Entropy of other attributes to get the information gain - **(i) Weather**:

$$\begin{aligned} H(S_{sunny}) &= -\left(\frac{1}{2}\right) \log_2 \left(\frac{1}{2}\right) - \left(\frac{1}{2}\right) \log_2 \left(\frac{1}{2}\right) \\ &= 1 \\ H(S_{rainy}) &= 0 \ (No\ Variation) \\ Conditional\ Entropy &= \sum_{x \in X} P(x) \cdot H(x) \\ &= P(S_{sunny}) \cdot H(S_{sunny}) + P(S_{rainy}) \cdot H(S_{rainy}) \\ &= \left(\frac{2}{4}\right) \cdot 1 + \left(\frac{2}{4}\right) \cdot 0 \\ &= 0.5 \\ Information\ Gain,\ IG(S, Weather) &= H(S) - H(S|Weather) \\ &= 0.8112 - 0.5 \\ &= 0.3112 \end{aligned}$$

(ii) Parents:

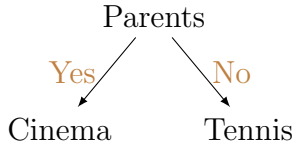
$$
\begin{aligned}
H(S_{Yes}) &= 0 \ (No\ Variation) \\
H(S_{No}) &= 0 \ (No\ Variation) \\
Conditional\ Entropy &= \sum_{x \in X} P(x) \cdot H(x) \\
&= P(S_{Yes}) \cdot H(S_{Yes}) + P(S_{No}) \cdot H(S_{No}) \\
&= \left(\frac{3}{4}\right) \cdot 0 + \left(\frac{1}{4}\right) \cdot 0 \\
&= 0 \\
Information\ Gain,\ IG(S, Weather) &= H(S) - H(S|Weather) \\
&= 0.8112 - 0 \\
&= \mathbf{0.8112}
\end{aligned}
$$

(iii) Money:

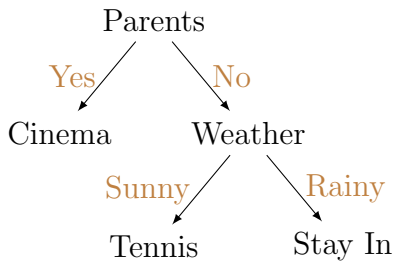
$$
\begin{aligned}
H(S_{poor}) &= 0 \ (No\ Variation) \\
H(S_{rich}) &= -\left(\frac{2}{3}\right) \log_2 \left(\frac{2}{3}\right) - \left(\frac{1}{3}\right) \log_2 \left(\frac{1}{3}\right) \\
&= 0.9182 \\
Conditional\ Entropy &= \sum_{x \in X} P(x) \cdot H(x) \\
&= P(S_{poor}) \cdot H(S_{poor}) + P(S_{rich}) \cdot H(S_{rich}) \\
&= \left(\frac{1}{4}\right) \cdot 0 + \left(\frac{3}{4}\right) \cdot 0.9182 \\
&= 0.6886 \\
Information\ Gain,\ IG(S, Weather) &= H(S) - H(S|Weather) \\
&= 0.8112 - 0.6886 \\
&= 0.1226
\end{aligned}
$$

The attribute **Parents** has the highest Information Gain as seen above, so it is chosen as the root node.

Now our decision tree is -



The final decision tree:



Part C: Customer Churn

(1) Customer Segmentation: Customer Segmentation refers to dividing customers into groups based on similarities so that they can be targeted and companies can market to each relevant group in an effective manner. By doing this, it is easier to analyze why certain groups tend to churn which helps to prevent similar new customers from leaving.

(2) Customer Churn: Customer churn refers to the number of customers that have stopped using a product or service over a given period of time. In our example, we are predicting the churn as the number of customers lost for Telco services. It is calculated using Churn rate which is the value obtained by dividing the number of customers lost during the given time period (ex: quarter) by the number of customers present at the beginning of that time period.

(3) Data Science approach to predict customer churn:
The K-Nearest Neighbors (KNN) model can be used to predict customer churn with upto 90% accuracy. This can be applied in the following manner -

- The new customer is taken as the data point to compare against other customers in the database who are most similar to them.
- Based on whether other similar customers churned or not, the model predicts the churn for the target customer.
- For a given dataset, we can take 70% into the training set and the remaining 30% into the testing set.
- Since KNN only works for numerical values, we need to remove the non-numerical attributes from the dataset.
- We can further optimize the model by improving the value of k (avoid overfitting) as well as the variables used (information gain) for the prediction.

References

- Wikipedia
- Towards Data Science
- Data Bricks
- IBM
- TCS
- Software Testing Help
- Geeks for Geeks
- Machine Learning Mastery