Module 3- Feature Generation and Feature Selection

Extracting Meaning from Data

Module 3 Syllabus	**Feature Generation and Feature Selection** **Extracting Meaning from Data:** Motivating application: user (customer) retention. Feature Generation (brainstorming, role of domain expertise, and place for imagination), Feature Selection algorithms. Filters; Wrappers; Decision Trees; Random Forests. Recommendation Systems: Building a User-Facing Data Product, Algorithmic ingredients of a Recommendation Engine, Dimensionality Reduction, Singular Value Decomposition, Principal Component Analysis, Exercise: build your own recommendation system.

Handouts for Session 1: Motivating Application: User Retention

3.1 Motivating application: user (customer) retention

- Suppose an app called Chasing Dragons charges a monthly subscription fee, with revenue increasing with more users.
- However, only 10% of new users return after the first month.
- **To boost revenue**, **there are two options**: **increase the retention rate of existing users** or **acquire new ones.**
- Generally, retaining existing customers is cheaper than acquiring new ones.
- Focusing on retention, a model could be built to predict if a new user will return next month based on their behavior this month.
- This model could help in providing targeted incentives, such as a free month, to users predicted to need extra encouragement to stay.
- **A good crude model:** Logistic Regression – Gives the probability the user returns their second month conditional on their activities in the first month.
- User behavior is recorded for the first 30 days after sign-up, logging every action with timestamps: for example, a user clicked "level 6" at 5:22 a.m., slew a dragon at 5:23 a.m., earned 22 points at 5:24 a.m., and was shown an ad at 5:25 a.m. This phase involves collecting data on every possible user action.
- User actions, ranging from thousands to just a few, are stored in timestamped event logs.
- These logs need to be processed into a dataset with rows representing users and columns representing features. This phase, known as **feature generation**, involves brainstorming potential features **without being selective**.

- The data science team, including game designers, software engineers, statisticians, and marketing experts, collaborates to identify relevant features.

  Here are some examples:

  - ✓ Number of days the user visited in the first month
  - ✓ Amount of time until second visit
  - ✓ Number of points on day j for j=1, . . .,30 (this would be 30 separate features)
  - ✓ Total number of points in first month (sum of the other features)
  - ✓ Did user fill out Chasing Dragons profile (binary 1 or 0)
  - ✓ Age and gender of user
  - ✓ Screen size of device
- Notice there are redundancies and correlations between these features; that's OK.

- To **construct a logistic regression model** predicting user return behavior, the initial focus lies in attaining a functional model before refinement. Irrespective of the subsequent time frame, classification ci=1 designates a returning user. The logistic regression formula targeted is:

$$\text{logit}\left(P\left(c_i = 1 \middle| x_i\right)\right) = \alpha + \beta^{\tau} \cdot x_i$$

- Initially, a comprehensive set of features is gathered, encompassing user behavior, demographics, and platform interactions.
- Following data collection, **feature subsets must be refined for optimal predictive power** during model scaling and production.
- Three main methods guide feature subset selection: **filters**, **wrappers**, and **embedded methods**.
- Filters independently evaluate feature relevance, wrappers use model performance to assess feature subsets, and embedded methods incorporate feature selection within model training.

Questions

1.Define Customer Retention?

2.What are different relevant features of the Chasing Dragon app?

3.How to boost the revenue of Chasing Dragon application.

Handouts for Session 2: Feature Generation or Feature Extraction

3.2 Feature Generation or Feature Extraction

Feature generation, **also known as feature extraction**, is the process of transforming raw data into a structured format ***where each column represents a specific characteristic or attribute (feature) of the data***, and ***each row represents an observation or instance***.

- This involves identifying, creating, and selecting meaningful variables from the raw data that can be used in machine learning models to make predictions or understand patterns.
- This process is both an art and a science. Having a domain expert involved is beneficial, but using creativity and imagination is equally important.
- Remember, **feature generation is constrained by two factors**: the feasibility of capturing certain information and the awareness to consider capturing it.

Information can be categorized into the following buckets:

- ***Relevant and useful, but it's impossible to capture it.***
  Keep in mind that much user information isn't captured, like free time, other apps, employment status, or insomnia, which might predict their return. Some captured data may act as proxies for these factors, such as playing the game at 3 a.m. indicating insomnia or night shifts.
- ***Relevant and useful, possible to log it, and you did.***
  The decision to log this information during the brainstorming session was crucial. However, mere logging doesn't guarantee understanding its relevance or usefulness. The feature selection process aims to uncover this information.
- ***Relevant and useful, possible to log it, but you didn't.***
  Human limitations can lead to overlooking crucial information, emphasizing the need for creative feature selection. Usability studies help identify key user actions for better feature capture.
- ***Not relevant or useful, but you don't know that and log it.***
  Feature selection aims to address this: while you've logged certain information, unknowing its necessity.
- ***Not relevant or useful, and you either can't capture it or it didn't occur to you.***

Feature Generation or Feature Extraction.

Questions:

1.Define Feature Generation.

2.Define Feature Extraction.

3.Define Feature Generation. Explain how information can be categorized in feature generation in detail.

Handouts for Session 3: Feature Selection: Filters and Wrappers

3.3 Feature Selection Algorithms

Feature selection involves identifying the most relevant and informative features from a dataset for building predictive models.

1. Filters:

- ✓ Filters prioritize features based on specific metrics or statistics, such as correlation with the outcome variable, offering a quick overview of predictive power of .
- ✓ However, ***they may ignore redundancy*** and ***fail to consider feature interactions***, potentially resulting in correlated features and limited insight into complex relationships.
- ✓ By treating the features as independent, it is not taking into account possible interactions or correlation between features.
- ✓ However in some cases 2 redundant features can be more powerful when used together and appear useless when considered individually.

2. Wrappers

- ✓ Wrapper feature selection explores subsets of features of a predetermined size, seeking to identify combinations that optimize model performance.
- ✓ However, as the number of potential combinations grows exponentially with the number of features.

  The number of possible size k subsets of n things, called $\binom{n}{k}$.
- ✓ This exponential growth of possible feature subsets can lead to overfitting.

- ✓ In **wrapper** feature selection, **two key aspects** require consideration:
- ✓ **first**, ***the choice of an algorithm for feature selection***, and
- ✓ **second**, ***the determination of a selection criterion or filter*** to find out the usefulness of the chosen feature set.

A. Selecting an algorithm

Stepwise regression is a category of feature selection methods which involves systematically adjusting feature sets within regression models, typically through **forward selection**, **backward elimination**, or **a combined approach**, to optimize model performance based on predefined selection criteria.

Forward selection:

Forward selection *involves* ***systematically adding features to a regression model one at a time*** ***based on*** ***their ability to improve model performance according to a selection criterion****. This iterative process* ***continues until further feature additions no longer enhance the model's performance****.*

Backward elimination:

Backward elimination *begins with* ***a regression model*** ***containing all features.*** *Subsequently,* ***one feature is systematically removed at a time,*** *the feature whose removal makes the biggest improvement in the selection criterion****. Stop removing features*** ***when removing the feature makes the selection criterion get worse.***

Combined approach:

The ***combined approach*** *in feature selection* ***blends forward selection and backward elimination*** *to strike a balance between* ***maximizing relevance*** ***and minimizing redundancy****. It* ***iteratively adds and removes features based on their significance and impact on model fit,*** *resulting in a subset of features optimized for predictive power.*

B. Selection criterion

The choice of selection criteria in feature selection methods may seem arbitrary. To address this, experimenting with various criteria can help assess model robustness. Different criteria may yield diverse models, necessitating the prioritization of optimization goals based on the problem context and objectives.

R-squared

R-squared can be interpreted as the proportion of variance explained by your model.

$$R^2 = 1 - \frac{\Sigma i \left(yi - \widehat{yi}\right)^2}{\Sigma i \left(yi - \overline{y}\right)^2}$$

p-values

In regression analysis, the interpretation of p-values involves assuming a null hypothesis where the coefficients (βs) are zero. A low p-value suggests that observing the data and obtaining the estimated coefficient under the null hypothesis is highly unlikely, indicating a high likelihood that the coefficient is non-zero.

AIC (Akaike Information Criterion)

Given by the formula **2k−2ln(L)** , where k is the number of parameters in the model and **ln(L)** is the "maximized value of the log likelihood." **The goal is to minimize AIC.**

BIC (Bayesian Information Criterion)

Given by the formula **k*ln(n) −2ln(L)**, where k is the number of parameters in the model, n is the number of observations (data points, or users), and ln(L) is the maximized value of the log likelihood. The goal is to minimize BIC.

Entropy

Entropy is a measure of disorder or impurity in the given dataset.

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Questions:

1.Define Feature Selection.

2.Explain Filter Method

3.Explain Wrapper Method.

4.Explain selecting an algorithm in wrapper method.

5.Explain the different Selecting Criteria in feature selection.

Handouts for Session 4: Decision tree

3. Embedded Methods: Decision Trees

i)Decision Trees

- ✓ Decision trees are a popular and powerful tool used in various fields such as machine learning, data mining, and statistics. They provide a clear and intuitive way to make decisions based on data by modelling the relationships between different variables.
- ✓ A decision tree is a flowchart-like structure used to make decisions or predictions. It consists of nodes representing decisions or tests on attributes, branches representing the outcome of these decisions, and leaf nodes representing final outcomes or predictions.
- ✓ Each internal node corresponds to a test on an attribute, each branch corresponds to the result of the test, and each leaf node corresponds to a class label or a continuous value. In the context of a data problem, a decision tree is a classification algorithm.
- ✓ For the Chasing Dragons example, you want to classify users as “Yes, going to come back next month” or “No, not going to come back next month.” This isn’t really a decision in the colloquial sense, so don’t let that throw you.
- ✓ You know that the class of any given user is dependent on many factors (number of dragons the user slew, their age, how many hours they already played the game). And you want to break it down based on the data you’ve collected. But how do you construct decision trees from data and what mathematical properties can you expect them to have?

- ✓ But you want this tree to be based on data and not just what you feel like. Choosing a feature to pick at each step is like playing the game 20 Questions really well. You take whatever the most informative thing is first. Let's formalize that—we need a notion of "informative."
- ✓ For the sake of this discussion, assume we break compound questions into multiple yes-or-no questions, and we denote the answers by "0" or "1." Given a random variable X, we denote by p(X)=1 and p(X) =0 the probability that X is true or false, respectively.

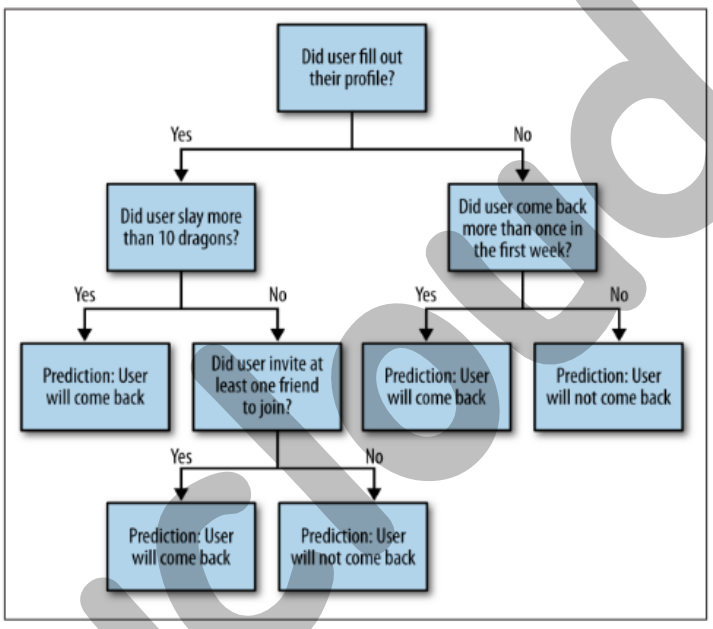


Figure 7-4. Decision tree for Chasing Dragons

Entropy

- Entropy is a measure of disorder or impurity in the given dataset.
- In the decision tree, messy data are split based on values of the feature vector associated with each data point.
- With each split, the data becomes more homogenous which will decrease the entropy. However, some data in some nodes will not be homogenous, where the entropy value will not be small.The higher the entropy, the harder it is to draw any conclusion.

- When the tree finally reaches the terminal or leaf node maximum purity is added.
- For a dataset that has C classes and the probability of randomly choosing data from class, i is Pi. Then entropy E(S) can be mathematically represented as

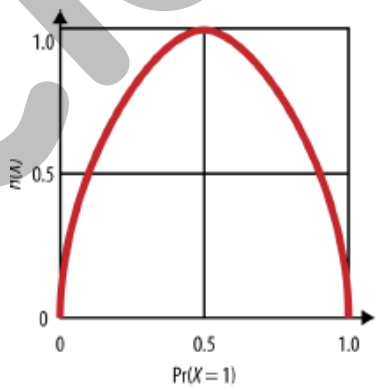
$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

- To quantify what is the most "informative" feature, we define entropy– effectively a measure for how mixed up something is—for X

$$H(X) = - p(X=1) \log_2(p(X=1)) - p(X=0) \log_2(p(X=0))$$

when p(X =1)= 0 or p (X =0) =0, the entropy vanishes i.e. if either option has probability zero, the entropy is 0(pure).As p(X =1) =1− p(X =0) , the entropy is symmetric about 0.5 and maximized at 0.5.

- In particular, if either option has probability zero, the entropy is 0. Moreover, because p (X =1) =1− p (X =0) , the entropy is symmetric about 0.5 and maximized at 0.5, which we can easily confirm using a bit of calculus. The Below Figure shows the picture of that.



- **Example:**

If we have a dataset of 10 observations belonging to two classes YES and NO. If 6 observations belong to the class, YES, and 4 observations belong to class NO, then entropy can be written as below.

$$E(S) = -(P_{yes}\log_2 P_{yes} + P_{no}\log_2 P_{no})$$

Pyes is the probability of choosing Yes and *Pno* is the probability of choosing a No. Here *Pyes* is 6/10 and *Pno* is 4/10.

$$E(S) = -(6/10 * \log_2 * 6/10 + 4/10 * \log_2 * 4/10) \approx 0.971$$

If all the 10 observations belong to 1 class then entropy will be equal to zero. Which implies the node is a pure node.

$$E(S) = -(1 \log_2 1) = 0$$

If both classes YES and NO have an equal number of observations, then entropy will be equal to 1.

$$E = -(\tfrac{5}{10} * \log_2 \tfrac{5}{10} + \tfrac{5}{10} * \log_2 \tfrac{5}{10}) = -2(0.5 \log_2 0.5) = 1$$

- Entropy is a measurement of how mixed up something is.If X denotes the event of a baby being born a boy, the expectation is to be true or false with probability close to 1/2, which corresponds to high entropy, i.e., the bag of babies from which we are selecting a baby is highly mixed.
- If X denotes the event of a rainfall in a desert, then it's low entropy.In other words, the bag of day-long weather events is not highly mixed in deserts.
- X is the target of our model. So, X could be the event that someone buys something on our site.
- Which attribute of the user will tell us the most information about this event X needs to be determined
- Information Gain, IG(X,a), for a given attribute a, is the entropy we lose(reduction in entropy, or uncertainty) if we know the value of that attribute
- IG (X,a) =H(X) − H (X|a).
- H(X|a) can be computed in 2 steps:
- For any actual value a_0 of the attribute a we can compute the specific conditional entropy H(X|$a = a_0$) as:
- H(X|$a = a_0$) = -p(X=1| $a = a_0$) log_2(p(X=1| $a = a_0$))
  - p(X=0| $a = a_0$) log_2(p(X=0| $a = a_0$))

  And then we can put it all together, for all possible values of a, to getthe conditional entropy H(X|a) :

  H(X|a)=$\sum_{a_i} p(a = a_i)$. H(X|$a = a_i$)

Decision Tree Algorithm

- The Decision Tree is built iteratively.

- It starts with the root. - You need an algorithm to decide which attribute to split on; e.g., which node should be the next one to identify.
- The attribute is chosen in order to maximize information gain.
- Keep going until all the points at the end are in the same class or you end up with no features left. In this case, you take the majority vote.
- The Tree can be pruned to avoid overfitting. - cutting it off below a certain depth.
- If you build the entire tree, it's often less accurate with new data than if you prune it.

- **Example:**

  - Suppose you have your Chasing Dragons dataset. Your outcome variable is Return: a binary variable that captures whether or not the user returns next month, and you have tons of predictors.

    # Load necessary library

    #Loads the rpart library, which is used for recursive partitioning and regression trees.

    library(rpart)

    # Read the CSV file

    chasingdragons <- read.csv("chasingdragons.csv")

    setwd("F:/college/data science-2021 scheme/dscience")

    # Grow the classification tree

    #Builds a classification tree model to predict the Return variable using the other variables (profile, num_dragons, num_friends_invited, gender, age, num_days) as predictors. The method="class" specifies that this is a classification tree.

    model1 <- rpart(Return ~ profile + num_dragons + num_friends_invited + gender + age + num_days, method="class", data=chasingdragons)

Display the results

#Prints the complexity parameter (CP) table, which helps in understanding the performance of the model and in pruning the tree.

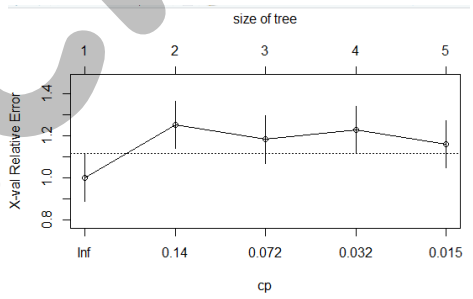
```
printcp(model1)
```

```
Variables actually used in tree construction:
[1] gender              num_days            num_friends_invited

Root node error: 44/100 = 0.44

n= 100

        CP nsplit rel error xerror    xstd
1 0.181818      0   1.00000 1.0000 0.11282
2 0.113636      1   0.81818 1.1591 0.11361
3 0.045455      2   0.70455 1.1136 0.11361
4 0.022727      3   0.65909 1.1136 0.11361
5 0.010000      4   0.63636 1.1818 0.11355
```

Visualize cross-validation results

plot of the cross-validation results for the complexity parameter. The plotcp function helps visualize how the model's error changes with the complexity of the tree, which aids in selecting the optimal tree size.

```
plotcp(model1)
```



Detailed summary of the model

#The summary function provides comprehensive information about the model, including splits, nodes.

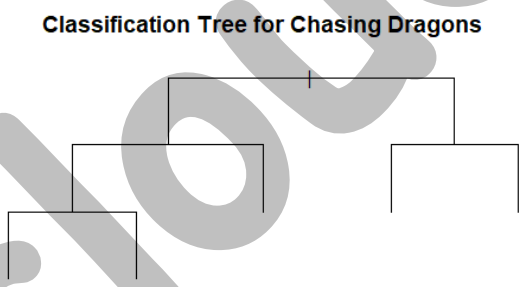
```
summary(model1)
```

#sample summary output:

```
Variable importance
num_friends_invited              gender            num_days                 age
                 41                  27                  15                   9
        num_dragons
                  7

Node number 1: 100 observations,    complexity param=0.1818182
  predicted class=No   expected loss=0.44  P(node) =1
    class counts:    56    44
   probabilities: 0.560 0.440
  left son=2 (62 obs) right son=3 (38 obs)
  Primary splits:
```

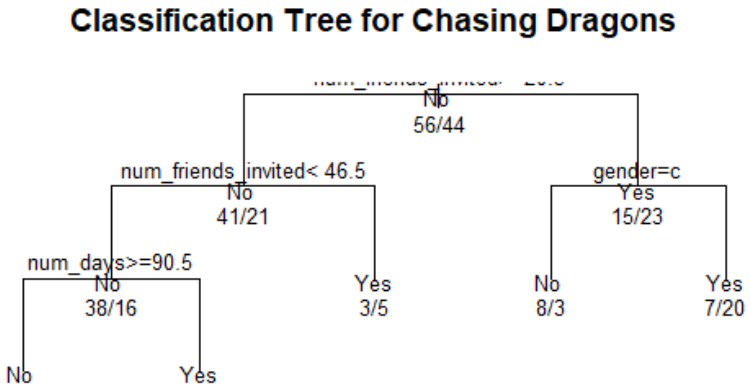
Plot the classification tree.The plot function visualizes the tree structure, and the uniform=TRUE argument makes the branch lengths uniform. The main argument specifies the title of the plot.

```
plot(model1, uniform=TRUE, main="Classification Tree for Chasing Dragons")
```



Add text labels to the tree plot. The text function annotates the tree plot with information about the nodes. The use.n=TRUE argument includes the number of observations at each node, all=TRUE ensures all nodes are labeled, and cex=.8 sets the size of the text labels.

```
text(model1, use.n=TRUE, all=TRUE, cex=.8)
```



Handling Continuous Variables

- In case of continuous variables, the correct threshold of a value needs to be determined to consider it as a binary variable.
- Example: A User's number of Dragon Slays can be partitioned into categories such as "less than 10" and "at least 10". Now we have a binary variable case.
- In this case, it takes some extra work to decide on the information gain because it depends on the threshold as well as the feature.
- Instead of a single threshold, bins of values can be created for the attribute. – Depends on situation

Questions:

1.Define Decision tree.

2.Explain Decision Tree for Chasing Dragons Problem.

3. Suppose you have your Chasing Dragons dataset. Your outcome variable is Return: a binary variable that captures whether or not the user returns next month, and you have tons of predictors. Write a R Script using decision tree algorithm for the above scenario.

4.Write Decision Algorithm in detail.

Handouts for Session 5: Random forest

ii)Random Forest

- Random forests generalize decision trees with bagging,otherwise known has Bootstrap Aggregating.

- Makes the models more accurate and more robust, but at the cost of interpretability
- But easy to specify – 2 Hyperparameters: Number of Trees (N) in the forest and Number of Features (F) to randomly select for each tree
- A bootstrap sample is a sample with replacement, which means we might sample the same data point more than once.We usually take to the sample size to be 80% of the size of the entire (training) dataset, but of course this parameter can be adjusted de-pending on circumstances. This is technically a third hyper parameter of our random forest algorithm.
- To construct a random forest, you construct N decision trees as follows:
  - For each tree, take a bootstrap sample of your data, and for each node you randomly select F features, say 5 out of the 100 total features.
  - Then you use your entropy-information-gain engine as described in the previous section to decide which among those features you will split your tree on at each stage.
- Note that you could decide beforehand how deep the tree should get,or you could prune your trees after the fact, but you typically don't prune the trees in random forests, because a great feature of random forests is that they can incorporate idiosyncratic noise.

- **Algorithm**
  - Select random K data points from the Training Set
  - Build a Decision Tree based on the selected K points
  - Select the Number of Trees to build and repeat Steps 1 & 2
  - For a new data point (test data point) makes each of the trees predict the class for the data point. And assign the new data point the average across all the predicted classes

Questions

1.Define Random Forest.

Explain Random forest Algorithm.

Handouts for Session 6: User Retention

3.4 User Retention: Interpretability Vs. Predictive Power

- Assume a model predicts well, but can you find the meaning in the model?
- Example: The prediction could be “the more the user plays in the first month, the more likely the user is to come back next month”. This is obvious and not very helpful when doing the analysis
- It could also tell that showing them ads in the first 5 minutes decreases their chances of coming back, buts its ok to show ads after the first hour. This would give an insight not to show ads in the first 5 minutes
- To study this more, you really would want to do some A/B testing, but this initial model and feature selection would help you prioritize the types of tests you might want to run.
- Features that are associated with the user’s behaviour are qualitatively different from the features associated with one’s own behaviour.
- If there’s a correlation of getting a high number of points in the first month with players returning to play next month, does that mean if you just give users a high number of points this month without them playing at all, they’ll come back – No!
- It’s not the number of points that caused them to come back, it’s that they’re really into playing the game which correlates with both their coming back and their getting a high number of points.
- Therefore, do feature selection with all variables, but then focus on the ones you can do something about conditional on user attributes.

Questions:

1.Explain the User Retention in Detail.

Handouts for Session 7: User Retention, Dimensionality Reduction, SVD

3.5 A Real-World Recommendation Engine

- Recommendation engines are used all the time—what movie would you like, knowing other movies you liked? What book would you like, keeping in mind past purchases?
- There are plenty of different ways to go about building such a model, but they have very similar feels if not implementation. To set up a recommendation engine, suppose you have users, which form a set U; and you have items to recommend, which form a set V.
- We can denote this as a bipartite graph (shown again in Figure below) if each user and each item has a node to represent it—there are lines from a user to an item if that user has expressed an opinion about that item.
- Note they might not always love that item, so the edges could have weights: they could be positive, negative, or on a continuous scale (or discontinuous, but many-valued like a star system).
- The implications of this choice can be heavy but we won't delve too deep here for us they are numeric ratings.
- Next up, you have training data in the form of some preferences—you know some of the opinions of some of the users on some of the items. From those training data, you want to predict other preferences for your users. That's essentially the output for a recommendation engine.
- You may also have metadata on users (i.e., they are male or female, etc.) or on items (the color of the product). For example, users come to your website and set up accounts, so you may know each user's gender, age, and preferences for up to three items.
- You represent a given user as a vector of features, sometimes including only metadata—sometimes including only preferences (which would lead to a sparse vector because you don't know all the user's opinions) and sometimes including both, depending on what you're doing with the vector.
- Also, you can sometimes bundle all the user vectors together to get a big user matrix, which we call U, through abuse of notation.

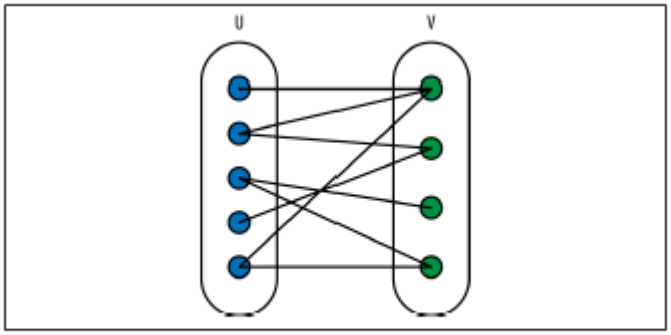


Figure 8-1. Bipartite graph with users and items (television shows) as nodes

3.6 The Dimensionality Problem

- As we increase the dimension (Number of features), the accuracy of the system increases up to a certain limit. Beyond the limit, the accuracy starts to decline.
- Solution - **Dimensionality Reduction:** This does not mean removing features, but rather transform the data into a different perspective.
- Our goal is to build a model that has a representation in a low dimensional subspace that gathers "taste information" to generate recommendations. So we're saying here that taste is latent but can be approximated by putting together all the observed information we do have about the user.

3.6.1 Singular Value Decomposition (SVD)

- **Rank of a Matrix:** The rank of a matrix A is the maximum number of linearly independent row vectors or column vectors in the matrix. It is denoted as rank(A). In case of SVD the rank of A is the number of non-zero singular values in the diagonal matrix S.
- **Latent Features:** These are the hidden patterns or factors in the data. For a user-item matrix, they represent underlying user preferences and item characteristics.
- Importance of Latent Features: The singular values in S indicate the importance of these features.
- A larger singular value means the corresponding latent feature is more significant in capturing the structure of the data.
- Given an m×n matrix X of rank k: $\mathbf{X=USV^T}$

- **U**: m×k matrix (Left Singular Vectors) that contains user latent features. Each row corresponds to a user, and each column represents a latent feature. For example, a row might capture a user's preference for genres like action or comedy.
- **S**: k×k diagonal matrix with singular values. These values indicate the importance of each latent feature. Larger values correspond to more significant features.
- **V:** k×n matrix (Right Singular Vectors) Contains item latent features. Each row corresponds to an item, and each column represents a latent feature. For example, a row might capture an item's characteristics like genre or popularity.
- **U and V:** The columns of U and V are orthogonal, meaning they capture independent latent features.
- **S:** The singular values in S measure the importance of each latent feature. Larger values indicate more significant features.

Dimensionality Reduction with SVD

To reduce the dimensionality, follow these steps:

1. Compute SVD:
    - Perform SVD on the matrix X to get U, S, and V.
2. Select Top d Singular Values:
    - Identify the top d largest singular values from S. These singular values correspond to the most significant components of the data.
3. Construct Reduced Matrices:
    - Construct reduced matrices U_d, S_d, and V_d
4. Approximate the Original Matrix:

$$X_d = U_d S_d V_d^T$$

- This X_d is the best approximation of X, using only the top d singular values.

In Singular Value Decomposition (SVD), dimensionality reduction is achieved by selecting the most significant singular values and their corresponding singular vectors.

This process leverages the properties of the SVD to capture the most important structures in the data while discarding less important information.

Questions:

1.Explain Real world recommendation engine with neat diagram.

2.What is Dimensionality Problem.

3.Explain SVD in detail

Handouts for Session 8: PCA,Building Recommendation Engine

3.6.2 Principal Component Analysis

- In this approach, we aim to predict preferences by factorizing the **user-item interaction matrix X** into **two lower-dimensional matrices, U and V**, without the need for the singular values matrix S. The goal is to find U and V such that:

$$X \approx U \cdot V^T$$

- The optimization problem is to **minimize the discrepancy between the actual user-item interaction matrix X and its approximation**

$$\tilde{X} = U \cdot V^T.$$

- This discrepancy is measured using the squared error:

$$\text{argmin} \sum_{i,j} (x_{ij} - u_i \cdot v_j)^2$$

- x_{ij} is the actual interaction (e.g., rating) between **user i** and **item j**.
- u_i is the i-th row of matrix U, representing the **latent features of user i.**
- v_j is the j-th row of matrix V, representing the **latent features of item j.**
- The **dot product** $u_i \cdot v_j$ is the **predicted preference of user i for item j.**

Latent Features and Matrix Dimensions

- **Number of Latent Features (d)**: This is a parameter that you choose, representing the number of latent features you want to use. It controls the dimensions of matrices U and V.
- **Matrix U**: Has dimensions m×d, where m is the number of users and d is the number of latent features. Each row corresponds to a user.

- **Matrix V**: Has dimensions n×d, where n is the number of items and d is the number of latent features. Each row corresponds to an item.

3.7 Alternating Least Squares

- Here we are not first minimizing the squared error and then minimizing the size of the entries of the matrices U and V. Here we are actually doing both at the same time.
- Alternating Least Squares (ALS) is an algorithm for matrix factorization. ALS is used to decompose a given user-item interaction matrix into two lower-dimensional matrices (U and V) that capture latent features of users and items.

- Here's the algorithm:

  Pick a random V.

  Optimize U while V is fixed.

  Optimize V while U is fixed.
- Keep doing the preceding two steps until you're not changing very much at all. To be precise, you choose an ϵ and if your coefficients are each changing by less than ϵ, then you declare your algorithm "converged."
- Fix V and Update U The way you do this optimization is user by user. So for user i, you want to find:

$$argmin_{u_i} \Sigma_{j \in P_i} (p_{i,j} - u_i^* v_j)^2$$

- where v_j is fixed. In other words, you just care about this user for now. But wait a minute, this is the same as linear least squares, and has a closed form solution! In other words, set:

$$u_i = \left(V_{*,j}^T V_{*,i}\right)^{-1} V_{*,j}^T P_{*i},$$

- where V*,i is the subset of V for which you have preferences coming from user i. Taking the inverse is easy because it's d×d, which is small.And there aren't that many

preferences per user, so solving this many times is really not that hard. Overall you've got a doable update for U.

- When you fix U and optimize V, it's analogous—you only ever have to consider the users that rated that movie, which may be pretty large for popular movies but on average isn't; but even so, you're only ever inverting a d×d matrix.

3.8 Building Recommendation System using Python

The following code is Matt's code to illustrate implementing a recommendation system on a relatively small dataset.

Initialize Matrix V:

- V is initialized with random values.
- This matrix represents the latent features of items.

Initialize Matrix U:

- U is initialized with zeros.
- This matrix will represent the latent features of users.

ALS Algorithm

The ALS algorithm alternates between updating the user latent features (U) and the item latent features (V) to minimize the squared error of the predicted ratings.

For each user:

- Extract the items they have interacted with and their corresponding ratings.
- **Create a matrix vo** containing the latent features of these items.
- **Create a vector pvo** of the ratings.

Solve the regularized least squares problem:

- Update U[i, :] using the formula:

$$U[i,:] = (V_i^T V_i + \lambda I)^{-1} V_i^T X_i$$

- `V_i` is the submatrix of `V` corresponding to the items user `i` has rated.
- `X_i` is the vector of ratings for these items.

3. **Regularized Least Squares Solution**:

   - The equation $U[i, :] = \left(V_i^T V_i + \lambda I\right)^{-1} V_i^T X_i$ is used to update `U[i, :]`.

Explanation of the Equation

- $V_i^T V_i$:
  - This is the product of the transpose of `V_i` and `V_i` itself. It captures the correlations between the latent features of the items that user `i` has interacted with.
- λI:
  - I is the identity matrix of size equal to the number of latent features (`num_features`). Multiplying it by the regularization parameter λ adds a small value to the diagonal elements, which helps in preventing overfitting and ensures numerical stability by making the matrix invertible.
- $\left(V_i^T V_i + \lambda I\right)^{-1}$:
  - This is the inverse of the regularized matrix. It essentially normalizes the correlations captured in $V_i^T V_i$.
- $V_i^T X_i$:
  - This is the product of the transpose of `V_i` and `X_i`. It projects the known interaction values into the latent feature space.
- **Combining these**:
  - The equation $U[i, :] = \left(V_i^T V_i + \lambda I\right)^{-1} V_i^T X_i$ provides the best fit for the latent feature vector of user `i` by solving the regularized least squares problem.

Error Calculation

- Calculate the root mean square error (RMSE):Compute the prediction error for all known user-item interactions.
- Final Predicted Matrix
- Predict the entire user-item interaction matrix:
- Multiply U and V.T to get the predicted ratings.

Code:

```
import math
import numpy as np
# Define the user-item ratings matrix pu
pu = [
   [(0, 0, 1), (0, 1, 22), (0, 2, 1), (0, 3, 1), (0, 5, 0)],
   [(1, 0, 1), (1, 1, 32), (1, 2, 0), (1, 3, 0), (1, 4, 1), (1, 5, 0)],
   [(2, 0, 0), (2, 1, 18), (2, 2, 1), (2, 3, 1), (2, 4, 0), (2, 5, 1)],
   [(3, 0, 1), (3, 1, 40), (3, 2, 1), (3, 3, 0), (3, 4, 0), (3, 5, 1)],
   [(4, 0, 0), (4, 1, 40), (4, 2, 0), (4, 4, 1), (4, 5, 0)],
   [(5, 0, 0), (5, 1, 25), (5, 2, 1), (5, 3, 1), (5, 4, 1)]
]
# Define the item-user ratings matrix pv
pv = [
   [(0, 0, 1), (0, 1, 1), (0, 2, 0), (0, 3, 1), (0, 4, 0), (0, 5, 0)],
   [(1, 0, 22), (1, 1, 32), (1, 2, 18), (1, 3, 40), (1, 4, 40), (1, 5, 25)],
   [(2, 0, 1), (2, 1, 0), (2, 2, 1), (2, 3, 1), (2, 4, 0), (2, 5, 1)],
   [(3, 0, 1), (3, 1, 0), (3, 2, 1), (3, 3, 0), (3, 5, 1)],
   [(4, 1, 1), (4, 2, 0), (4, 3, 0), (4, 4, 1), (4, 5, 1)],
   [(5, 0, 0), (5, 1, 0), (5, 2, 1), (5, 3, 1), (5, 4, 0)]
```

```
]
# Define matrix V
V = np.mat([
    [0.15968384, 0.9441198, 0.83651085],
    [0.73573009, 0.24906915, 0.85338239],
    [0.25605814, 0.6990532, 0.50900407],
    [0.2405843, 0.31848888, 0.60233653],
    [0.24237479, 0.15293281, 0.22240255],
    [0.03943766, 0.19287528, 0.95094265]
])
# Initialize matrix U with zeros
U = np.mat(np.zeros([6, 3]))
# Regularization parameter
L = 0.03


# Perform matrix factorization using alternating least squares
for iter in range(5):
    print("\n----- ITER %s -----" % (iter + 1))
    print("U")
    urs = []


    # Update U
    for uset in pu:
        vo = []
```

```
        pvo = []
        for i, j, p in uset:
            vor = []
            for k in range(3):
                vor.append(V[j, k])
            vo.append(vor)
            pvo.append(p)

        vo = np.mat(vo)
    ur = np.linalg.inv(vo.T * vo + L * np.mat(np.eye(3))) * vo.T * np.mat(pvo).T
        urs.append(ur.T)

    U = np.vstack(urs)
    print(U)

print("V")
    vrs = []

    # Update V
    for vset in pv:
        uo = []
        puo = []
        for j, i, p in vset:
            uor = []
```

```
            for k in range(3):
                uor.append(U[i, k])
            uo.append(uor)
            puo.append(p)

        uo = np.mat(uo)
        vr = np.linalg.inv(uo.T * uo + L * np.mat(np.eye(3))) * uo.T * np.mat(puo).T
        vrs.append(vr.T)

    V = np.vstack(vrs)
    print(V)
# Calculate RMSE (Root Mean Squared Error)
    err = 0.
    n = 0.
    for uset in pu:
        for i, j, p in uset:
            err += (p - (U[i] * V[j].T)[0, 0]) ** 2
            n += 1

    rmse = math.sqrt(err / n)
    print("RMSE:", rmse)

# Print final U * V.T
print("\nFinal U * V.T")
```

print(U * V.T)

Output:

```
----- ITER 1 -----
U
[[ 27.48856092  -6.39423002   0.76339065]
 [ 36.40898688 -10.96084483   2.19757677]
 [ 19.79884149  -6.62335704   2.23966925]
 [ 44.36457118 -13.91334026   3.61771668]
 [ 46.25668408 -15.11954225   4.75538378]
 [ 24.08497085 -11.99211139   7.2307236 ]]
V
[[ 0.0545325   0.11205172 -0.0332261 ]
 [ 0.4588277  -1.51294995 -0.61387609]
 [ 0.19340572  0.7878437   0.78829   ]
 [ 0.25513015  1.06866178  1.06871888]
 [-0.16891975 -0.7083464  -0.47721002]
 [-0.01489877 -0.04766114  0.12516857]]
0.458261949227786

----- ITER 2 -----
U
[[ 21.24745657  -9.63073906   5.11174585]
 [ 30.23955333 -13.58612407   5.88615241]
 [ 16.56669002  -8.71313163   5.6304904 ]
 [ 38.39283553 -17.04724985   8.01303322]
 [ 36.73866642 -17.32271228   7.40352948]
 [ 21.68875723 -12.66751199   8.14720285]]
V
[[ 0.28051193  0.75036922  0.39661661]
 [ 0.50058836 -1.50093297 -0.59823823]
 [ 0.19972091  0.76013758  0.77453081]
 [ 0.15257727  0.71636732  0.83603449]
 [-0.28061042 -0.89379554 -0.53418132]
 [ 0.05743832  0.32749297  0.48997755]]
0.23261838473656263

----- ITER 3 -----
U
[[ 20.03935133  -9.53918778   5.11576922]
 [ 28.24667145 -13.626873     5.99360507]
 [ 15.41955317  -8.73439494   5.7077762 ]
 [ 35.92368317 -17.13471274   8.32390394]
 [ 34.53667699 -17.24112282   7.36684371]
 [ 20.7631954  -12.19828589   7.49059224]]
V
[[ 0.33533083  0.82398023  0.39028879]
 [ 0.54163697 -1.47535262 -0.56710963]
 [ 0.17854914  0.68262722  0.74839685]
 [ 0.09531085  0.5462863   0.75891467]
 [-0.33339151 -0.98162282 -0.56601839]
 [-0.00402394  0.17911791  0.45039733]]
0.22364368105565205
```

```
U
[[ 19.11565545  -9.36172458   4.96588607]
 [ 26.88500489 -13.38916067   5.67368401]
 [ 14.72121272  -8.51670509   5.42125844]
 [ 34.07500028 -16.94324729   8.14133866]
 [ 32.96525324 -16.89010525   6.88873846]
 [ 19.91764225 -11.81949685   6.94254806]]
V
[[ 3.77508344e-01  8.80568062e-01  3.91782977e-01]
 [ 5.81916099e-01 -1.45004580e+00 -5.38150854e-01]
 [ 1.53422966e-01  5.97339864e-01  7.18293461e-01]
 [ 6.57958249e-02  4.59551458e-01  7.32442207e-01]
 [-3.78045460e-01 -1.05294790e+00 -5.92358399e-01]
 [-8.14718299e-02 -5.40684446e-04  4.05487216e-01]]
0.21919522056558113

----- ITER 5 -----
U
[[ 18.37609424  -9.14504116   4.75023946]
 [ 25.79448179 -13.10478426   5.28381689]
 [ 14.15580472  -8.27945838   5.08762866]
 [ 32.58674446 -16.65571486   7.77775649]
 [ 31.69560339 -16.47980863   6.29323664]
 [ 19.18869022 -11.48480019   6.45928646]]
V
[[ 0.40861253  0.91893055  0.39447878]
 [ 0.62155035 -1.42528866 -0.51155646]
 [ 0.12949727  0.5172765   0.68830999]
 [ 0.05454709  0.42318187  0.73517698]
 [-0.41861857 -1.11464585 -0.61380526]
 [-0.16925641 -0.19571672  0.3603905 ]]
0.21484614865128004
 [[ 0.9789133  22.02597562  0.91877642  0.62461344 -0.41481413  0.391507  ]
  [ 0.58191546 32.00769898  0.19842198 -0.25415279  0.56590965  0.10318157]
  [ 0.18295347 17.99655423  1.05223443  1.00874867  0.17996792  1.05800087]
  [ 1.07806674 40.01474243  0.95779204  0.44714287  0.14977915  0.54731624]
  [ 0.28996936 39.96955196 -0.08842598 -0.61841076  1.23796031  0.12871294]
  [-0.16494307 24.99160291  0.99005714  0.93524659  0.80399875  1.32782419]]
```

Questions:

1.Explain PCA in detail.

2.Define Alternating Least Sqaure.

3.Write a program for Recommendation system using Python.