adaptTo()

EUROPE'S LEADING AEM DEVELOPER CONFERENCE

27th – 29th SEPTEMBER 2021

Content Based Recommendation Engine in AEM

Ankit Gubrani, PlayStation

About the Speaker

Ankit Gubrani

Sr. Software Engineer @ PlayStation

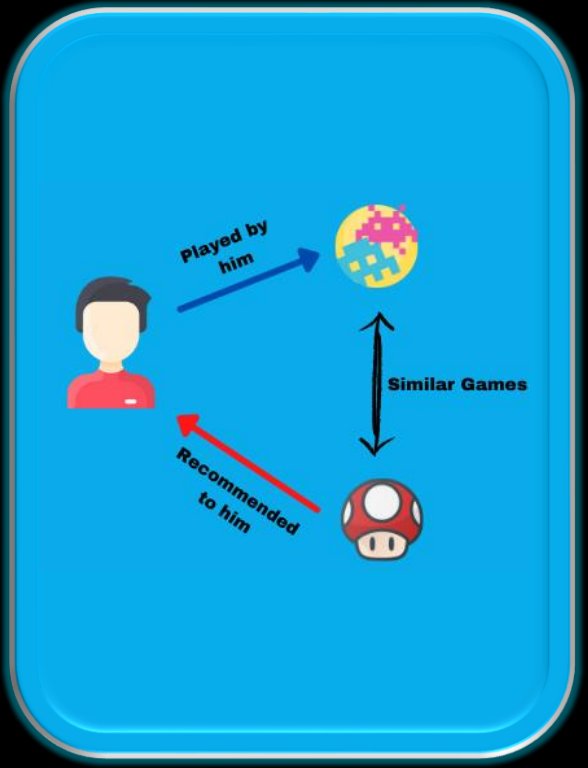
https://www.linkedin.com/in/ankitgubrani/

Recommendation Engines

What are Recommendation Engines?

- Data filtering tools which generate personalized recommendations of content or products
- Generates recommendations using mathematical algorithms & data available
- Improves user experience & user retention rate

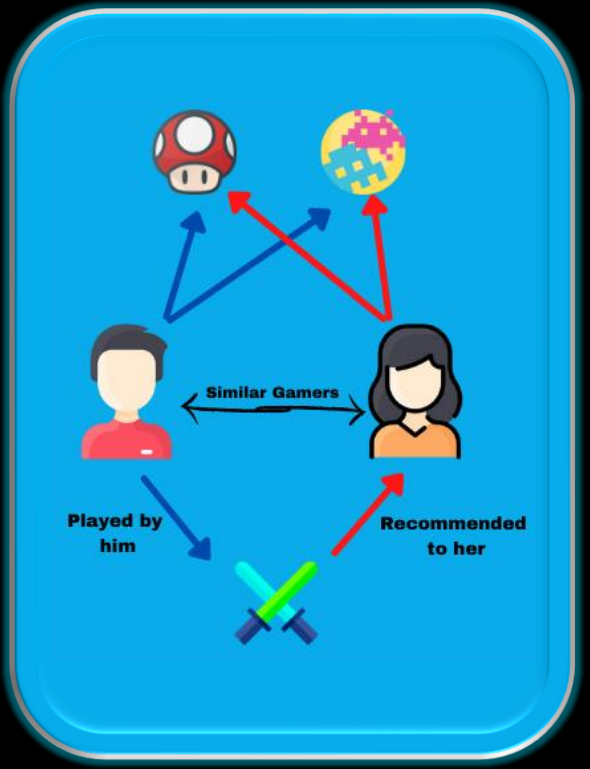
Types of Recommendation Engines



Content Based Filtering

- Technique in which only properties or keywords of items/products are considered while generating recommendations
- It is like recommending similar items

Types of Recommendation Engines



Collaborative Filtering

- Technique in which user behavior & products attributes are considered simultaneously to generate recommendations
- It is like “Other people also bought”

adaptTo()

Generating Recommendations in AEM

adaptTo()

Generating Recommendations in AEM

- Extract the Data
- Convert the properties/keywords extracted from data into vectors
- Find the cosine similarity between vectors representing items/products

Data Extraction

adaptTo()

Data Extraction

- AEM query requesting data for generating the data set
- Identify the properties important for the recommendations like description, rating tags, page type tags etc
- Generate the Bag of words from identified list of properties

Data Extraction

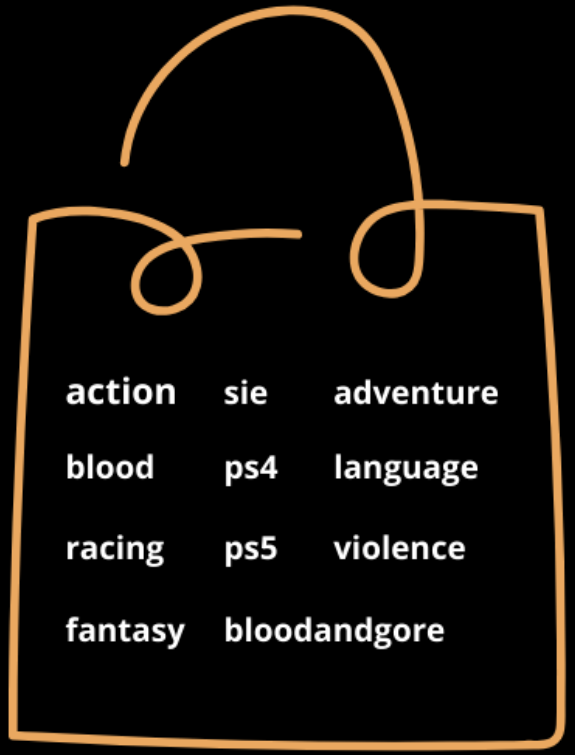
Node ID	Bag of Words/Features
_content_site_en-us_games_gt4	**driving racing sie ps4**
_content_site_en-us_games_lou	**action adventure sie ps4 violence stronglanguage**
_content_site_en-us_games_gow	**action sie ps4 bloodandgore violence stronglanguage**
_content_site_en-us_games_horizon	**action roleplaying sie ps4 language violence**
_content_site_en-us_games_astro	**action sie PS5 fantasy**

adaptTo()

Feature Extraction

- Computers have hard time directly working with raw text directly
- Bag-of-words model is a way of extracting features

Bag of words model



- Bag-of-words involves:
  - A vocabulary of all known words
  - And a measure of the presence of known words

Vectorize the Data

adaptTo()

Vectorize the Data

- Fundamental idea is to convert the textual information in the form of properties/keywords into machine readable format
- We will use Count Vectorizer to vectorize the data

Count Vectorizer Example

```
Vocabulary = ["Action", "Racing", "7plus", "Online Play",
              "Blood", "Fantasy", "Voilence"]
```

Racing Game = ["Racing", "7plus", "Online Play", "Racing"]

Action	Racing	Blood	7plus	Online Play	Fantasy	Voilence
0	2	0	1	1	0	0

Count Vectorizer

```java
public RealMatrix getCountMatrix(Collection<String> documents) {

    int rowDimensions = documents.size();
    int colDimensions = this.dictionary.getTotalTerms();

    // OpenMapRealMatrix can be initialized only if rowDimensions & colDimensions both are 1 or greater
    if (rowDimensions < 1 || colDimensions < 1) {
        return null;
    }

    RealMatrix matrix = new OpenMapRealMatrix(rowDimensions, colDimensions);

    int counter = 0;

    for (String document : documents) {
        matrix.setRowVector(counter++, this.getCountVector(document));
    }

    return matrix;
}

public RealVector getCountVector(String document) {
    RealVector vector = new OpenMapRealVector(this.dictionary.getTotalTerms());
    String[] tokens = tokenizer.getTokens(document);

    for (String token : tokens) {
        // Getting Index of the token from Dictionary
        Integer tokenIndex = this.dictionary.getTermIndex(token);

        if (tokenIndex != null) {
            vector.addToEntry(tokenIndex, 1);
        }
    }
    return vector;
}
```

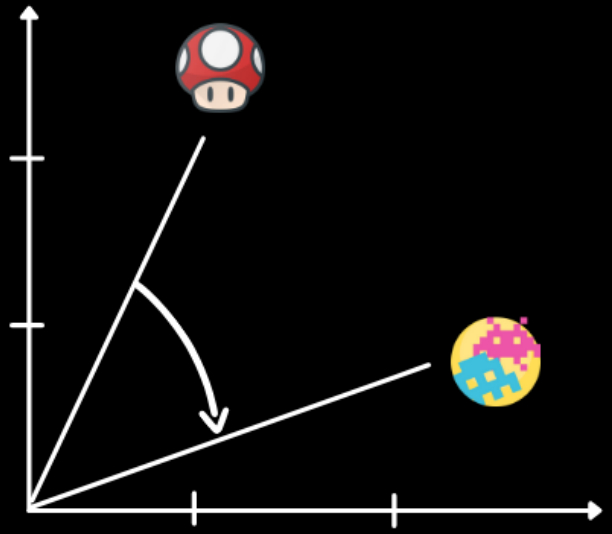
- Converts a collection of properties into a vector of terms
- Count Vectorizer converts set of strings into frequency representation.

adaptTo()

Cosine Similarity

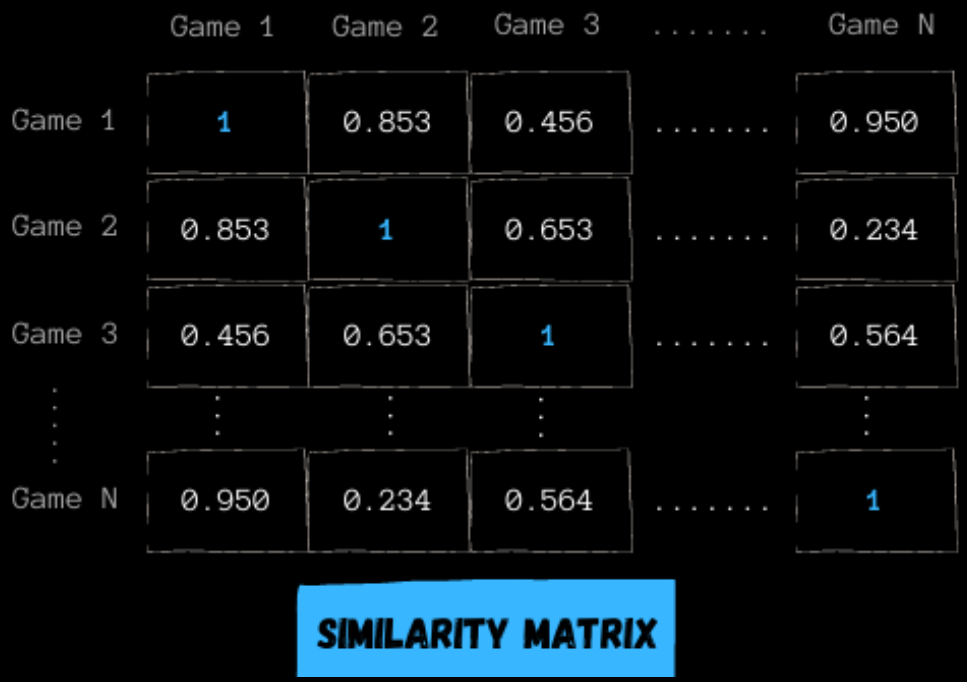
Cosine Similarity

$$\text{Similarity} = \cos(\Theta) = \frac{u \cdot v}{||u||\,||v||}$$



- Cosine Similarity is a measure of similarity between two vectors
- Cosine Similarity of 2 vectors is ratio between dot products of vectors & product of their magnitudes

Similarity Matrix



- Similarity on diagonals is 1, as every game is identical to itself
- Similarity Matrix is identical

Similarity Matrix - Snippet

```
// Initializing CountVectorizer object
CountVectorizer vectorizer = new CountVectorizer(dictionary, tokenizer);
// getting Matrix storing SparseVector for each Node/Data Element
RealMatrix countMatrix = vectorizer.getCountMatrix(bagOfWordsMap.values());
RealMatrix dotProductMatrix = new OpenMapRealMatrix(countMatrix.getRowDimension(), countMatrix.getRowDimension());

for (int row = 0; row < countMatrix.getRowDimension(); row++) {
    RealVector node1Vector = countMatrix.getRowVector(row);
    for (int col = 0; col < countMatrix.getRowDimension(); col++) {
        RealVector node2Vector = countMatrix.getRowVector(col);

        double cosine = 0.0D;

        /*  Calculating the cosine of the angle between this vector and the argument only if
            L2 norm (root of the sum of the squared elements) for both vectors is greater than zero. Because
            cosine is calculated as ::> DOT PRODUCTS OF VECTOR / (L2 norm of V1 * L2 norm of V2) */
        if (node1Vector.getNorm() > 0.0D && node2Vector.getNorm() > 0.0D) {
            cosine = node1Vector.cosine(node2Vector);
        }

        dotProductMatrix.setEntry(row, col, cosine);
    }
}
```

Consuming Recommendations

Consuming Recommendations - Snippet

```java
@Model(adaptables = {Resource.class, SlingHttpServletRequest.class}, defaultInjectionStrategy =
DefaultInjectionStrategy.OPTIONAL)
public class RecommendationsComponentModel {

  private RecommendationsReaderService recommendationsReaderService;

  @SlingObject
  private SlingHttpServletRequest slingRequest;

  @ScriptVariable
  private Page currentPage;

  @ValueMapValue
  @Default(values = StringUtils.EMPTY)
  private String recommendationEngineName;

  @ValueMapValue
  @Default(intValues = 3)
  private int numberOfRecommendations;

  private List<Resource> topRecommendations;

  @Inject
  public RecommendationsComponentModel(final RecommendationsReaderService recommendationsReaderService) {
      this.recommendationsReaderService = recommendationsReaderService;
  }

  @PostConstruct
  protected void init() {
      String nodePath = currentPage.getContentResource().getPath();
      topRecommendations  =
              recommendationsReaderService.getTopRecommendationsAsResouce(recommendationEngineName,
                              numberOfRecommendations, nodePath, slingRequest.getResourceResolver());
  }

  public List<Resource> getTopRecommendations() {
          return topRecommendations;
  }
}
```

- Service for reading the serialized & stored similarity matrix from JCR

Demo

adaptTo()

References

- Content-based filtering
- Bag of words model
- Cosine Similarity
- CountVectorizer
- Apache Commons Math
- AEM Implementation

Q&A

Thank you