Identification of TV Shows

Kajetan KACZMAREK
Nikhil SETHIA
Soham ROY

12 December, 2016 - 8 January, 2017

1 Introduction

1.1 Aim of the project

This project is about utilising the data mining skills learnt during lecture sessions to a real life problem. We have a very huge data set which consists of some sort of collection of dialogues from the top 300 IMDB shows.

- The main motive is to design a program which could let the client/end-user **identify which show** a random line belongs to.
- It can also be used in the future by the client to **generate new lines** and check if it suits a particular show.

1.2 Approach followed

- The initial data as viewed using the *KNIME* tool, directed us towards some filtering and data manipulation processes. This was due to the presence of a lot of unnecessary punctuations, symbols, time stamps and web-links.
- We then generated documents corresponding to each string which were later used in generating the data, which we could use for further learning, clustering and finally use for our main purpose, that is for prediction.

The following sections consist of a detailed discussion of our *KNIME* workflow, split into sections by different functions. Finally we have consolidated all the obtained results henceforth, along with the discussion on the correctness of methods. The *KNIME* workflow and other required files are also submitted along with this report.

2 Pre-processing

We need to pre-process the data before we do any operations on it. The text is formatted in the standard txt way, i.e. in rows. We need to bunch all them into a single cell, and **remove all the non-descriptive words**. This is because the provided data couldn't be directly utilised as it has been provided to us. We utilise some **row filtering** and **column filtering** operations for this purpose.

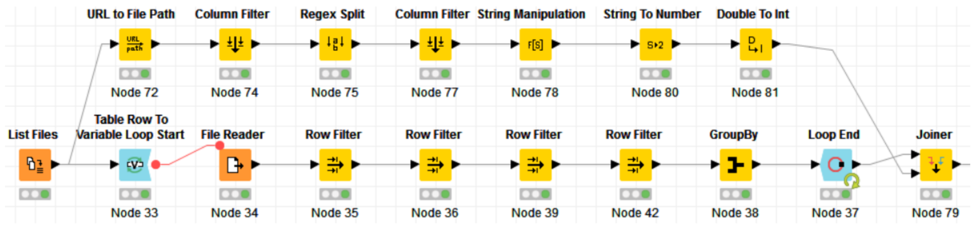


Figure 1: Pre-processing part of the workflow

- **List Files (Node 30)**: Used for listing all subtitle files that we would use for pre-processing
  - Makes a list of URLs of all files contained within a specified *location/path*
  - The *location/path* is specified as that of the topmost directory that contains all subtitle files
  - Configured to **Include Sub-folders** in order to encompass the subtitle files of all shows
- **Table Row To Variable Loop Start (Node 33)**: Creates a new iteration in the loop
  - Each URL is used to define a new variable value for each loop iteration
  - Names of variables are defined by the column names - in this case, the *location/path*
  - Configure to **Omit**, i.e. ignore missing values or empty table columns.
- **File Reader (Node 34)**: Reads data from file specified by URL location
  - The *Data URL* option is set to **URL** under the *Flow Variable* tab
  - Knime requires any arbitrary valid URL to be specified under the *Settings* tab
- **Row Filter**: Manipulates the row in order to eliminate *outliers* or *redundant data*
  - **Node 35**:
    - Removes all lines containing *timestamps* of the form *12:34:56,789* → *12:34:56,789* in the file.
    - Excludes rows using *regular expression [0-9]*:[0-9]*:[0-9]*,[0-9]** → *[0-9]*:[0-9]*:[0-9]*,[0-9]**
  - **Node 36**:
    - Removes all lines containing *dialogue numbers* of the form *12 34* in the file
    - Excludes rows using *regular expression [0-9]* **
  - **Node 39**:
    - Removes all lines containing information about the *corrector/syncer* of the file, which is usually indicated by a different *font* colour
    - Excludes rows using *regular expression* $^{<}$*font.**
  - **Node 42**: Removes all blank lines
- **GroupBy (Node 38)**:
  - Usually used for grouping the rows of a table by the unique values in the selected group columns.
  - The only column in our data is added on the *Manual Aggregation* tab, so it is *not* considered for type/pattern matching on other tabs.
  - All rows obtained after filtering are combined into one single row using the **Concatenate** method.

- **LoopEnd (Node 37)**:
  - Marks the end of the workflow loop and collects the intermediate results by row-wise concatenation
  - We get a table *Concatenate*(1)* with each row corresponding to one subtitles file.
- **URL to File Path (Node 72)**: Converts the URL strings from List Files into file path strings. Four columns are appended:
  1. Complete file paths
  2. Parent folder of the files
  3. File names (without extensions)
  4. File extensions
- **Columnn Filter (Node 74)**: Keeps only the *Parent folder* and *File Name* columns, and removes the rest.
- **Regex Split (Node 75)**: Used for splitting the file names into useful and non-useful parts.
  - Splits the string content of *Parent folder* column into two parts, where the latter part *split_1* contains the name of the show to which the file belongs.
  - The regular expression $(.*)\backslash\backslash(.*\$)$ is used
- **Columnn Filter (Node 77)**: Keeps only the useful *spit_1* column, and removes the rest.
- **String Manipulation (Node 78)**:
  - Manipulates strings of the *split_1* column
  - Appends to each tuple, the column *Iteration* of type *string* which contains the current iteration number in the loop.
- **String To Number (Node 80)**: Converts the entries of *Iteration* column from strings to numbers
- **Double To Int (Node 81)**: Converts the entries of *Iteration* column from double to integer.
- **Joiner (Node 79)**:
  - Performs *inner join* on the tables obtained from nodes *Loop End* and *Double to Int.*
  - We get 3 columns: *Concatenate*(1)*, *Iteration* and *split_1*

3 Document processing

We store every string into a separate document. This allows it to be used for further txt processing - removing punctuation marks, filtering out numbers/very small words, making everything small case (for uniformity) and filtering out certain kinds of words.

- **Strings To Document (Node 40)**:
  - Converts the each row of string type from the column *Concatenate*(1)* and stores it in a document.
  - Configured so that the subtitles for one TV show are all stored in one document
  - The name of the document is set to *split_1*, which allows us to know the name of the show.

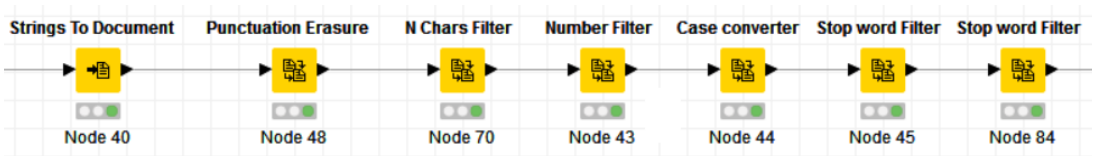


Figure 2: Document processing part of the workflow

  - The column *Document*, containing the names of documents, is appended to the output from *Joiner*.

- **Punctuation Erasure (Node 48)**: Removes punctuation characters in the documents fed to it.
- **N Chars Filter (Node 70)**: Configured to filter all terms with less than 3 characters.
- **Number Filter (Node 43)**: Filters digits, decimal separators "," or "." and possible leading "+" or "-".
- **Case converter (Node 44)**: Configured to converts everything to lower case.
- **Stop word Filter**:
  - **Nodes 45**: Configured to filter all words from a *built-in list* of English words.
  - **Nodes 84**: Configured to filter all words from a stop word list in the *specified* file.

4 Generating the Data

We implement the *bag of words* algorithm on the processed documents, in order to create a set of classes which would later be used by the prediction algorithms.

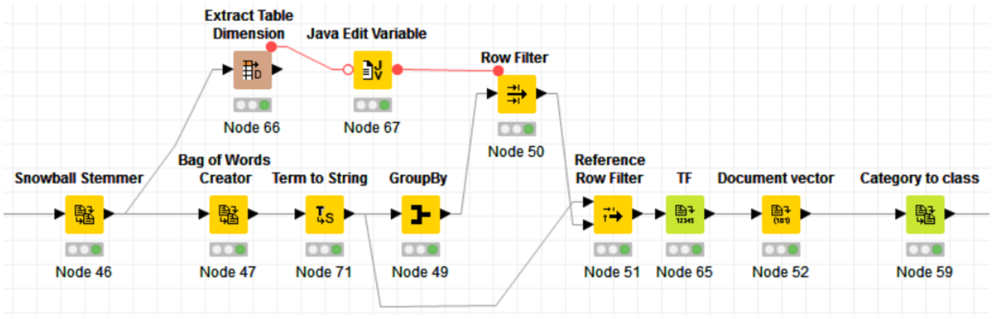


Figure 3: Generating data to be processed using the workflow

- **Snowball Stemmer (Node 46)**: Stems terms with the *Snowball stemming library*.
- **Bag of Words Creator (Node 47)**: Bag of Words model is a simplifying representation where a text is represented as a **bag (multi-set)** of its words, disregarding grammar and even word order but keeping multiplicity.
  - Creates a **BoW** (bag of words) for each document
  - BoW contains a column *Term*, containing the terms occurring in the corresponding document.

- **Term to String (Node 71)**:
  - Converts terms of column *Term* into strings
  - Attaches a new column *Term as String* containing these strings
  - Tags of the terms get lost, since only words of terms are converted
- **GroupBy (Node 49)**: Configured to count the number of documents in which each term occurs.
- **Extract Table Dimension (Node 66)**: Extracts the number of documents
- **Java Edit Variable (Node 67)**: Contains a Java code to compute the minimum document frequency
- **Row filter (Node 50)**: Configured to allow only terms/features which are 3 or more in number
- **Reference Row Filter (Node 51)**:
  - Configured to filter rows from the table fed by the *Term to String* node, using the table fed by the *Row Filter* as a **reference table**
  - Effectively filters the BoW generated by *Bag of Words Creator*
  - Configured to include rows from the reference table in its output
- **TF (Node 65)**: Computes the relative term frequency (tf) of each term according to each document and adds a column containing the tf value
- **Document vector (Node 52)**:
  - Configured to create a **binary document vector** for each document representing it in the **terms space**.
  - Only the column which corresponds to the document to which the row belongs, contains the value 1. The other columns contain the value 0
- **Category to class (Node 59)**: Adds a *Document class* column which contains its category as a string

5 Prediction

- **Color Manager (Node 55)**: Configured to assigns colours for nominal columns based on class

5.1 Decision Tree/K Nearest Neighbour/SVM

- **Column Filter (Node 58)**: Configured to filter out columns *Document* and those corresponding to the show *30rock*.
- **Partitioning (Node 57)**:
  - Configured to partition the *train and test data* with 70% size assigned to *training data*
  - Performs **stratified sampling** on the column *Document class*
- **Decision Tree Learner (Node 82)**:
  - Configured to induce a **classification decision tree** using **Gini index** as the *quality measure.*

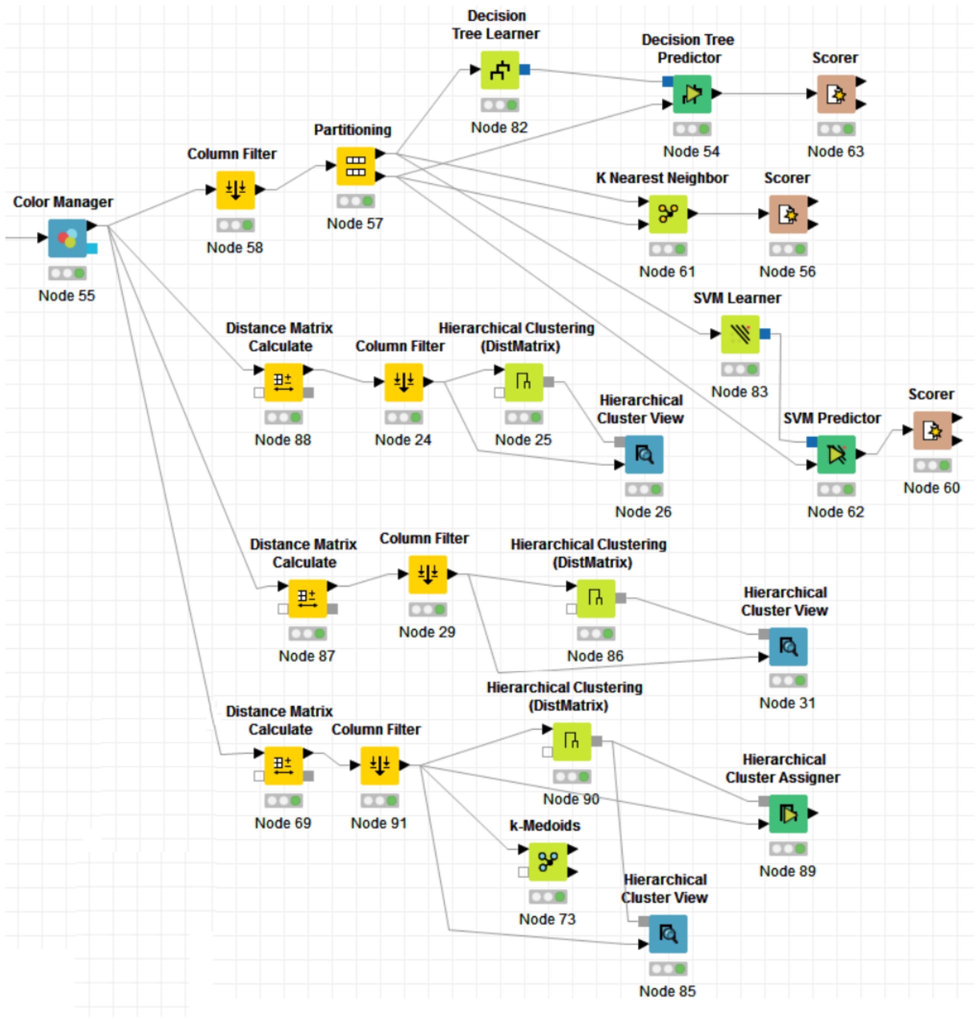


Figure 4: Prediction using various methods in the workflow

  - The algorithm is configured to run on 4 threads to exploit multiple processors/cores.

- **Decision Tree Predictor (Node 54)**:
  - Predicts the class value for new patterns using the decision tree
  - Configured to keep the maximum number of patterns for HiLite-ing as 10,000
- **K Nearest Neighbor (Node 61)**:
  - Classifies a set of test data based on the k Nearest Neighbor algorithm applied on the training data
  - Configured with *k=5* and neighbours are *weighted* by distance
- **SVM Learner (Node 83)**:
  - Trains a **support vector machine** on the *Document class* column
  - Configured to **polynomial kernel** and **overlapping penalty** of 1

- **SVM Predictor (Node 62)**:
  - Predicts the output for given values using the SVM model
  - Only the columns that have been used during training are used
  - The output table contains one extra column *Prediction (Document class)*, which contains the prediction
- **Scorer**: Shows the **confusion matrix**, i.e. how many rows of which attribute and their classification match
  - **Node 63**: Congifured to compare the columns *Document class* and *Prediction (Document class)* from *Decision Tree Predictor*
  - **Node 56**: Congifured to compare the columns *Document class* and *Class [kNN]*
  - **Node 60**: Congifured to compare the columns *Document class* and *Class [kNN]* from *SVM Predictor*

5.2 Hierarchical Clustering

- **Distance Matrix Calculate**: Calculates the distance values for all pairs of rows, appended as a single column *Distance* containing distance vector values
  - **Node 88**: Uses **Manhattan distance**
  - **Node 87**: Uses **Euclidean distance**
  - **Node 69**: Uses **Cosine distance**
- **Column Filter (Nodes 24, 29 and 91)**: Filters columns to keep only the columns *Document*, *Document class* and *Distance*
- **Hierarchical Clustering (DistMatrix) (Nodes 25, 86 and 90)**:
  - Hierarchically clusters the input data using a distance matrix.
  - Configured with linkage type as **Complete Linkage**
  - Defines distance between 2 clusters *c1* and *c2* as the maximal distance between any 2 points $x \in c1$ and $y \in c2$
- **Hierarchical Cluster Assigner (Node 89)**:
  - Configured to assign clusters based on **distance threshold** as 0.89
  - All those clusters in the dendogram are used with smallest distance to the threshold but below it
- **k-Medoids (Node 73)**:
  - Applies **k-Medoids algorithm** on the table from *Column Filter*
  - Configured with **k=2** and **chunk size = 1000**, i.e. the number of rows to consider at once
- **Hierarchical Cluster View (Nodes 26, 31 and 85)**:
  - Takes the table from *Column filter* and its **hierarchical cluster tree**
  - Visualizes the **cluster dendrogram** and a **distance plot** over all created levels

6 Results

We used all widely used classification methods. The results were surprisingly similar among all of them, so we would like to present all of the applied solutions

6.1 Hierarchical Cluster Views

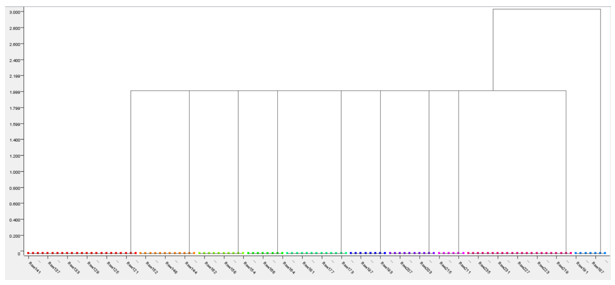


Figure 5: Dendogram obtained with Manhattan distance

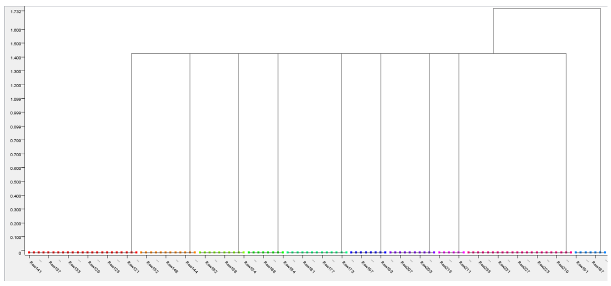


Figure 6: Dendogram obtained with Euclidean distance

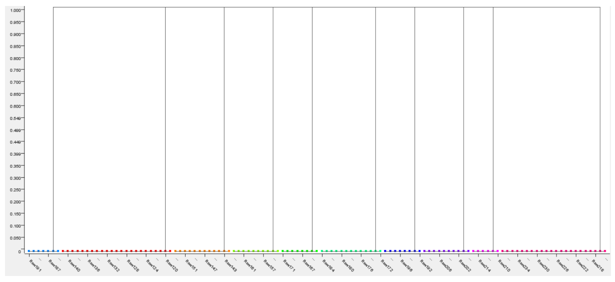
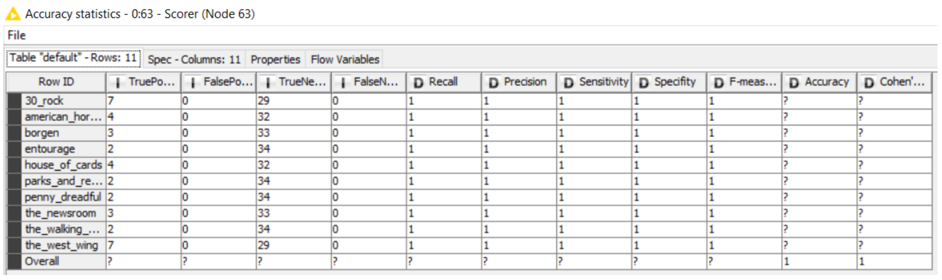


Figure 7: Dendogram obtained with Cosine distance

6.2 Accuracy of prediction



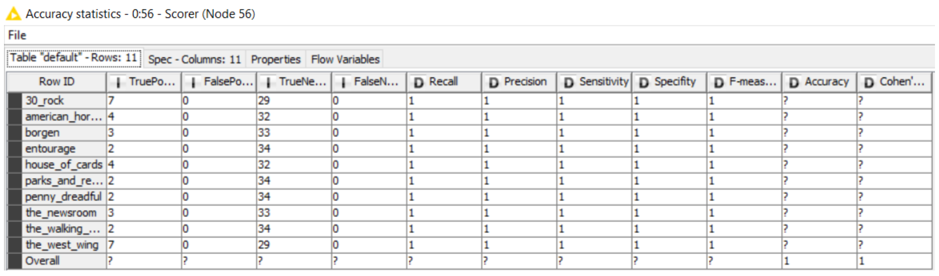
Accuracy statistics - 0:63 - Scorer (Node 63)

File

Table "default" - Rows: 11 | Spec - Columns: 11 | Properties | Flow Variables

Row ID	TruePo...	FalsePo...	TrueNe...	FalseN...	Recall	Precision	Sensitivity	Specifity	F-meas...	Accuracy	Cohen'...
30_rock	7	0	29	0	1	1	1	1	1	?	?
american_hor...	4	0	32	0	1	1	1	1	1	?	?
borgen	3	0	33	0	1	1	1	1	1	?	?
entourage	2	0	34	0	1	1	1	1	1	?	?
house_of_cards	4	0	32	0	1	1	1	1	1	?	?
parks_and_re...	2	0	34	0	1	1	1	1	1	?	?
penny_dreadful	2	0	34	0	1	1	1	1	1	?	?
the_newsroom	3	0	33	0	1	1	1	1	1	?	?
the_walking_...	2	0	34	0	1	1	1	1	1	?	?
the_west_wing	7	0	29	0	1	1	1	1	1	?	?
Overall	?	?	?	?	?	?	?	?	?	1	1

Figure 8: Accuracy of Decision Tree Predictor



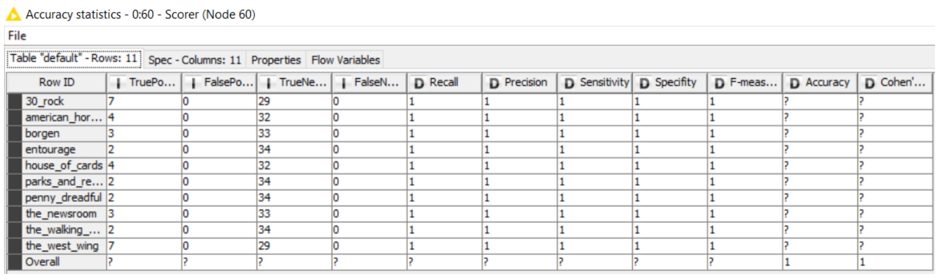
Accuracy statistics - 0:56 - Scorer (Node 56)

File

Table "default" - Rows: 11 | Spec - Columns: 11 | Properties | Flow Variables

Row ID	TruePo...	FalsePo...	TrueNe...	FalseN...	Recall	Precision	Sensitivity	Specifity	F-meas...	Accuracy	Cohen'...
30_rock	7	0	29	0	1	1	1	1	1	?	?
american_hor...	4	0	32	0	1	1	1	1	1	?	?
borgen	3	0	33	0	1	1	1	1	1	?	?
entourage	2	0	34	0	1	1	1	1	1	?	?
house_of_cards	4	0	32	0	1	1	1	1	1	?	?
parks_and_re...	2	0	34	0	1	1	1	1	1	?	?
penny_dreadful	2	0	34	0	1	1	1	1	1	?	?
the_newsroom	3	0	33	0	1	1	1	1	1	?	?
the_walking_...	2	0	34	0	1	1	1	1	1	?	?
the_west_wing	7	0	29	0	1	1	1	1	1	?	?
Overall	?	?	?	?	?	?	?	?	?	1	1

Figure 9: Accuracy of K Nearest Neighbor



Accuracy statistics - 0:60 - Scorer (Node 60)

File

Table "default" - Rows: 11 | Spec - Columns: 11 | Properties | Flow Variables

Row ID	TruePo...	FalsePo...	TrueNe...	FalseN...	Recall	Precision	Sensitivity	Specifity	F-meas...	Accuracy	Cohen'...
30_rock	7	0	29	0	1	1	1	1	1	?	?
american_hor...	4	0	32	0	1	1	1	1	1	?	?
borgen	3	0	33	0	1	1	1	1	1	?	?
entourage	2	0	34	0	1	1	1	1	1	?	?
house_of_cards	4	0	32	0	1	1	1	1	1	?	?
parks_and_re...	2	0	34	0	1	1	1	1	1	?	?
penny_dreadful	2	0	34	0	1	1	1	1	1	?	?
the_newsroom	3	0	33	0	1	1	1	1	1	?	?
the_walking_...	2	0	34	0	1	1	1	1	1	?	?
the_west_wing	7	0	29	0	1	1	1	1	1	?	?
Overall	?	?	?	?	?	?	?	?	?	1	1

Figure 10: Accuracy of SVM Learner

6.3 Problems

- **Lot of keywords specific to each show**
  - Troubled the clustering and the classification
  - For example: the names of the characters, the titles of shows etc.
  - We didn't find a way to remove those automatically, so we had to create a **separate Stop List**
  - The process of creating the Stop List was tedious so we may have overlooked certain words
  - For future projects, an **automated solution** needs to be developed.

- **Limited data processing ability:** We tried running the workflow on bigger subset of data, but it proved to require too much operating power, so we were forced to constrain ourselves to the small set

6.4 Conclusion

In the end, we managed to have a pretty good algorithm that was able to detect the programs with high accuracy. We still had to manually insert a lot of **keywords** to be omitted, so this is not a solution that can be used outside the **controlled environment**. If this problem is solved, we may have an automatic process for identifying the show to which a random line belongs.