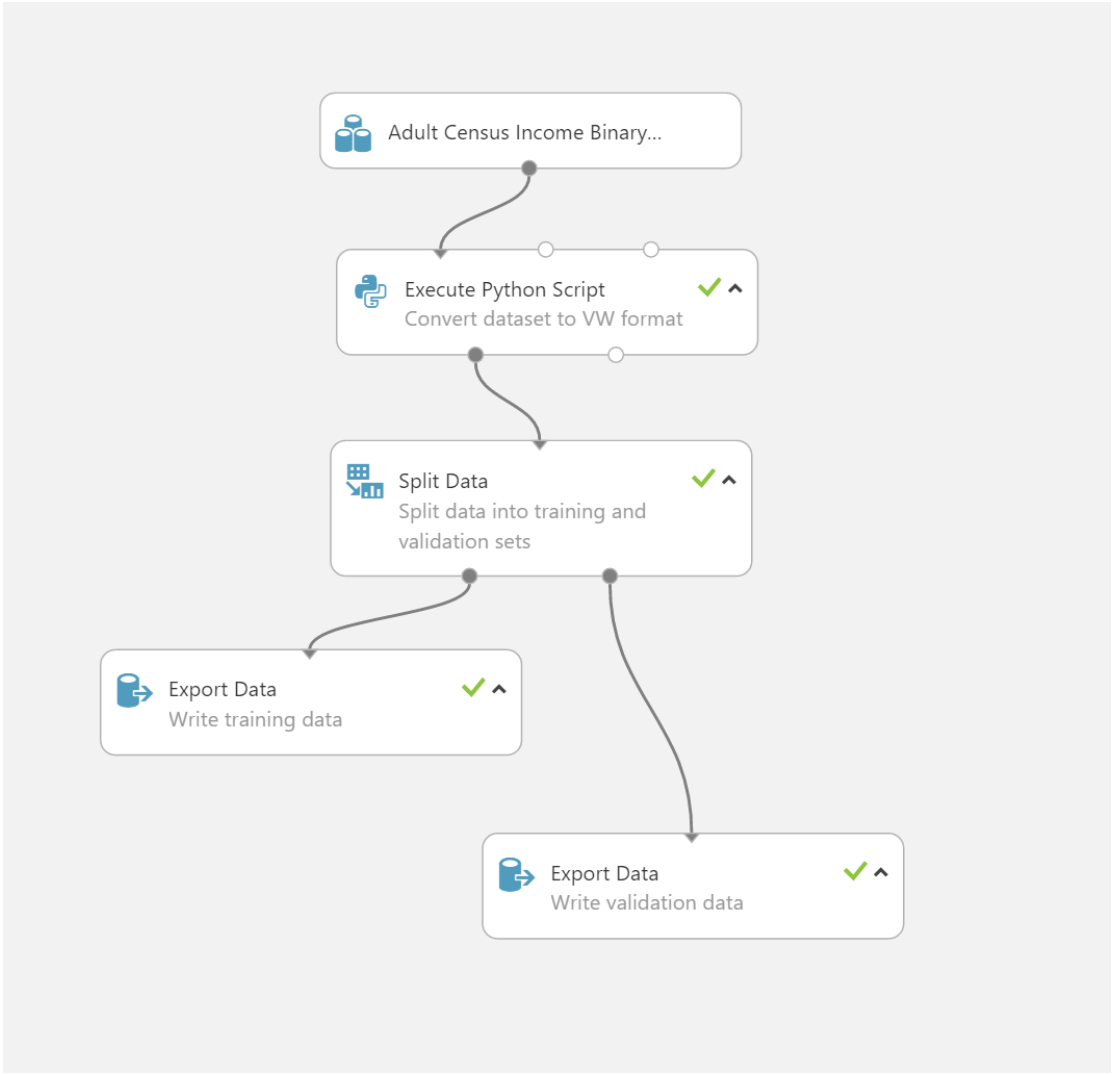
1. Azure Machine Learning with VW

STEP ONE:



Step 1. Convert Dataset to VW Format ❯ Adult Census Income Binary Classification dataset ❯ dataset

rows 32561 columns 15

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
view as															
	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K
	49	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in-family	Black	Female	0	0	16	Jamaica	<=50K
	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	45	United-States	>50K
	31	Private	45781	Masters	14	Never-married	Prof-specialty	Not-in-family	White	Female	14084	0	50	United-States	>50K
	42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	5178	0	40	United-States	>50K
	37	Private	280464	Some-college	10	Married-civ-spouse	Exec-managerial	Husband	Black	Male	0	0	80	United-States	>50K

Python Script:

Python script

```python
# convert a dataframe into VW format
import pandas as pd
import numpy as np


def azureml_main(inputDF):
    labelColName = 'income'
    trueLabel = '>50K'
    colsToExclude = ['workclass', 'occupation', 'native-country']
    numericCols = ['fnlwgt']

    output = convertDataFrameToVWFormat(inputDF, labelColName, trueLabel, colsToExclude, numericCols)
    return output

def convertDataFrameToVWFormat(inputDF, labelColName, trueLabel, colsToExclude, numericCols):
    # remove '|' and ':' that are special characters in VW
    def clean(s):
        return "".join(s.split()).replace("|", "").replace(":", "")

    def parseRow(row):
        line = []
        # convert labels to 1s and -1s and add to the beginning of the line
        line.append("{} |".format('1' if row[labelColName] == trueLabel else '-1'))

        for colName in featureCols:
            if (colName in numericCols):
                # format numeric features
                line.append("{}:{}".format(colName, row[colName]))
            else:
                # format string features
                line.append(clean(str(row[colName])))

        vw_line = " ".join(line)
        return vw_line

    # drop columns we don't need
    inputDF.drop(colsToExclude, axis = 1)

    # select feature columns
    featureCols = [c for c in inputDF.columns if c != labelColName]

    # parse each row
    output = inputDF.apply(parseRow , axis = 1).to_frame()
    return output
```

Split Data to training data:

Step 1. Convert Dataset to VW Format › Split Data › Results dataset1

rows	columns
22793	1

0

view as

-1 | 34 Private fnlwgt:148291 HS-grad 9 Married-civ-spouse Tech-support Wife White Female 0 0 32 United-States

-1 | 38 Private fnlwgt:223433 HS-grad 9 Never-married Machine-op-inspct Not-in-family White Male 0 0 40 United-States

-1 | 44 Private fnlwgt:159580 12th 8 Divorced Transport-moving Not-in-family White Female 0 0 40 United-States

Split data to validation data:

Step 1. Convert Dataset to VW Format › Split Data › Results dataset2

rows	columns
9768	1

0

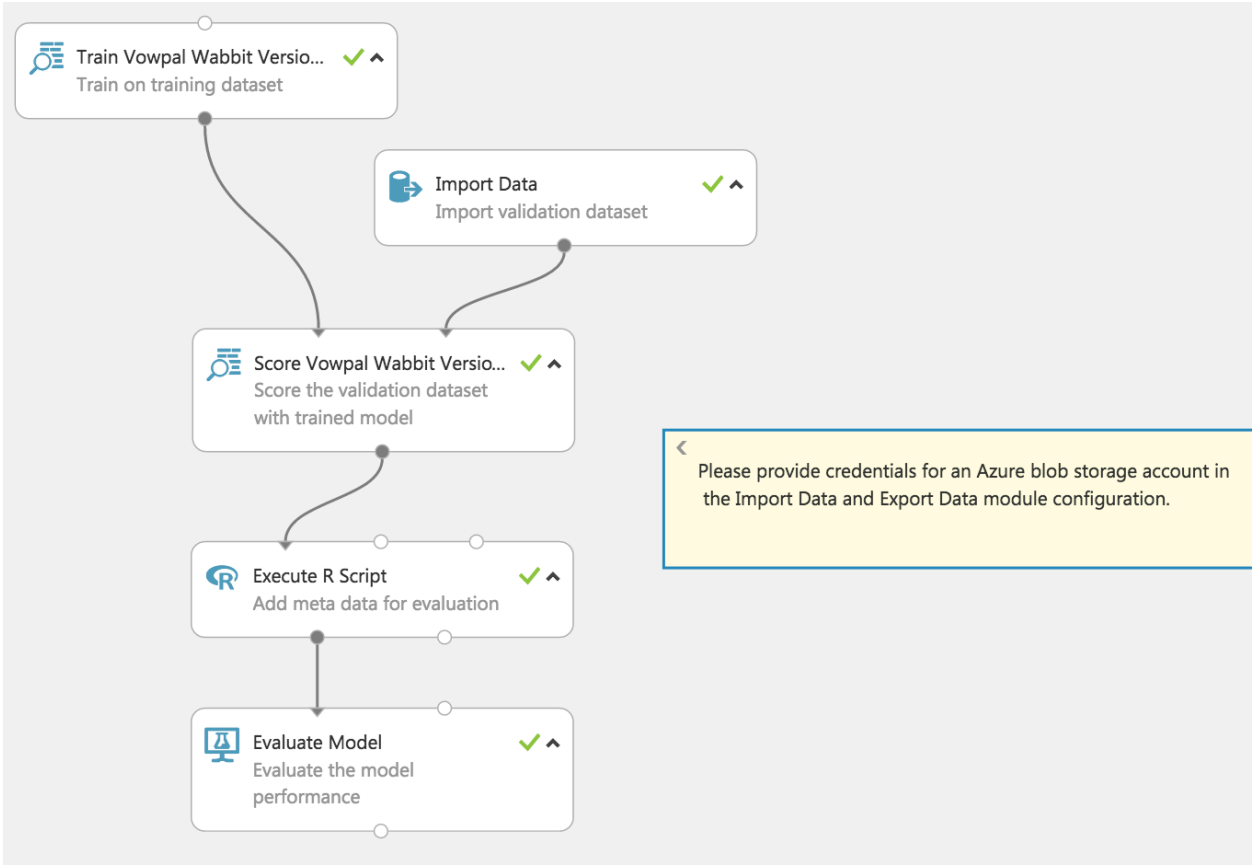
view as

-1 | 41 Private fnlwgt:187336 HS-grad 9 Separated Adm-clerical Unmarried White Female 0 0 40 United-States

1 | 30 Private fnlwgt:213722 Some-college 10 Never-married Sales Not-in-family White Male 0 0 50 United-States

1 | 64 Federal-gov fnlwgt:301383 Assoc-acdm 12 Married-civ-spouse Exec-managerial Husband White Male 9386 0 45 United-States

STEP TWO: train a VW model



Data are stored in blob in azure.

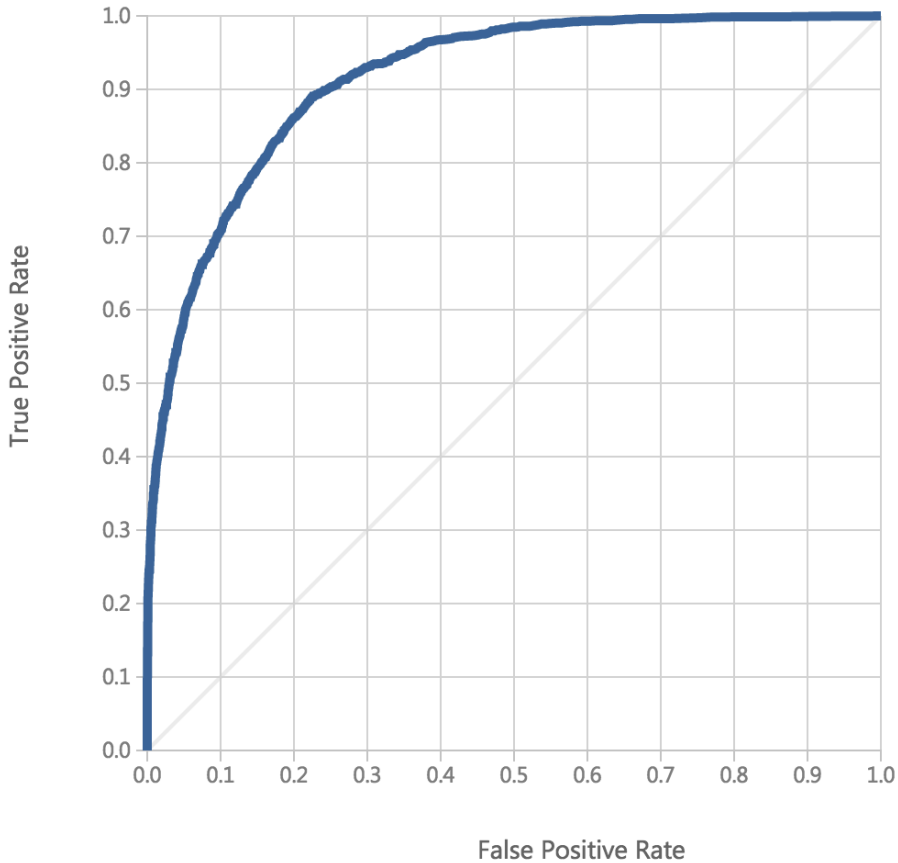
Here are the R code to calculate final score based on threshold:

```r
dataset <- maml.mapInputPort(1) # class: data.frame

# set the threshold
threshold = 0.5

# set negative class
dataset$MyScoredLabels[dataset$Results < threshold] <- -1

# set positive class
dataset$MyScoredLabels[dataset$Results >= threshold] <- 1

# Result is the probability when "--link logistic" is on. Rename it to MyScoredProbabilities
names(dataset)[names(dataset) == "Results"] <- "MyScoredProbabilities"

# set metadata for the scored probability and labels columns
dataset <- set.binary.classification.scores(dataset, "MyScoredProbabilities", "MyScoredLabels")

# set metadata for the label column
dataset <- set.true.label(dataset, "Labels")

maml.mapOutputPort("dataset");
```

Evaluate the model just created, we get a ROC graph and the AUC is 0.915.

Based on the standard, it is great discriminability, which shows this model is great.

ROC PRECISION/RECALL LIFT



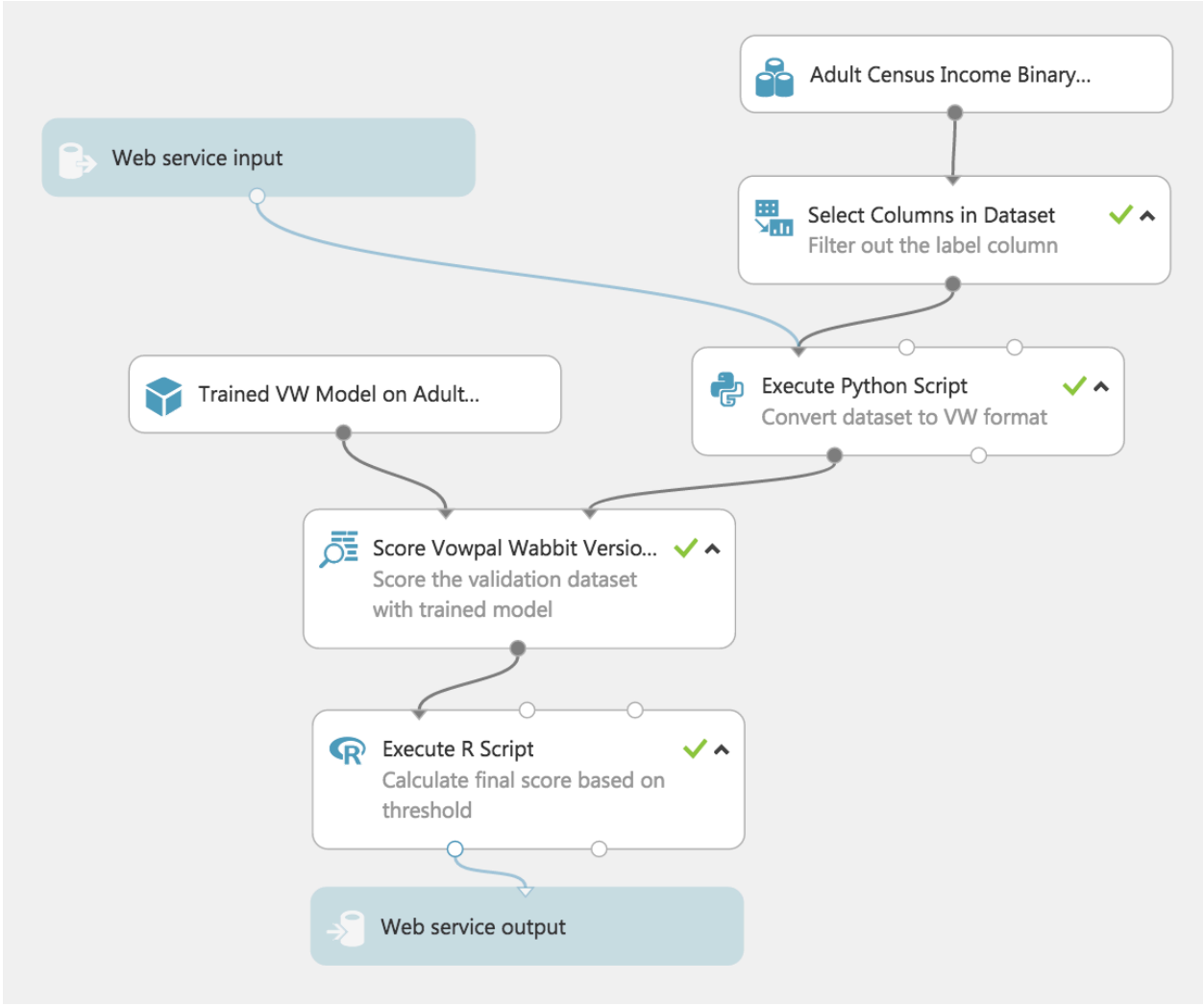
True Positive	False Negative	Accuracy	Precision	Threshold	AUC
1385	1018	0.860	0.799	0.5	0.915

False Positive	True Negative	Recall	F1 Score
349	7016	0.576	0.670

Positive Label	Negative Label
1	-1

Score Bin	Positive Examples	Negative Examples	Fraction Above Threshold	Accuracy	F1 Score	Precision	Recall	Negative Precision	Negative Recall	Cumulative AUC
(0.900,1.000]	393	5	0.041	0.794	0.281	0.987	0.164	0.785	0.999	0.000
(0.800,0.900]	199	14	0.063	0.813	0.393	0.969	0.246	0.802	0.997	0.000
(0.700,0.800]	288	60	0.098	0.836	0.523	0.918	0.366	0.827	0.989	0.003
(0.600,0.700]	258	116	0.136	0.851	0.609	0.854	0.474	0.850	0.974	0.010
(0.500,0.600]	247	154	0.178	0.860	0.670	0.799	0.576	0.873	0.953	0.021
(0.400,0.500]	243	282	0.231	0.856	0.698	0.721	0.677	0.897	0.914	0.045
(0.300,0.400]	213	346	0.288	0.842	0.705	0.653	0.766	0.919	0.867	0.079
(0.200,0.300]	256	581	0.374	0.809	0.692	0.574	0.873	0.950	0.788	0.144
(0.100,0.200]	195	1092	0.506	0.717	0.624	0.464	0.954	0.977	0.640	0.281
(0.000,0.100]	111	4715	1.000	0.246	0.395	0.246	1.000	1.000	0.000	0.915

STEP THREE: Operationalize a VW model



we can give an input, when run the model, the input data first be converted into VW format and then calculate a score, finally calculate the score we get to the final version based on threshold as outputs.

File_train.py

```
priti@priti-VirtualBox: ~
import csv
import re

location_train = '/home/priti/Downloads/train.tsv'
location_test = '/home/priti/Downloads/test.tsv'

location_train_vw = "rotten.train.vw" #will be created
location_test_vw = "rotten.test.vw" #will be created

#cleans a string "I'm a string!?" returns as "i m a string"
def clean(s):
  return " ".join(re.findall(r'\w+', s,flags = re.UNICODE | re.LOCALE)).lower()

#creates Vowpal Wabbit-formatted file from tsv file
def to_vw(location_input_file, location_output_file, test = False):
  print "\nReading:",location_input_file,"\nWriting:",location_output_file
  with open(location_input_file) as infile, open(location_output_file, "wb") as
outfile:
    #create a reader to read train file
    reader = csv.DictReader(infile, delimiter="\t")
    #for every line
    for row in reader:
      #if test set label doesnt matter/or isnt available
      if test:
        label = "1"
      else:
        label = str(int(row['Sentiment'])+1)
      phrase = clean(row['Phrase'])
      outfile.write(    label +
          " '"+row['PhraseId'] +
          " |f " +
          phrase +
          " |a " +
          "word_count:"+str(phrase.count(" ")+1)
          + "\n" )

to_vw(location_train, location_train_vw)
to_vw(location_test, location_test_vw, test=True)
priti@priti-VirtualBox:~$
```

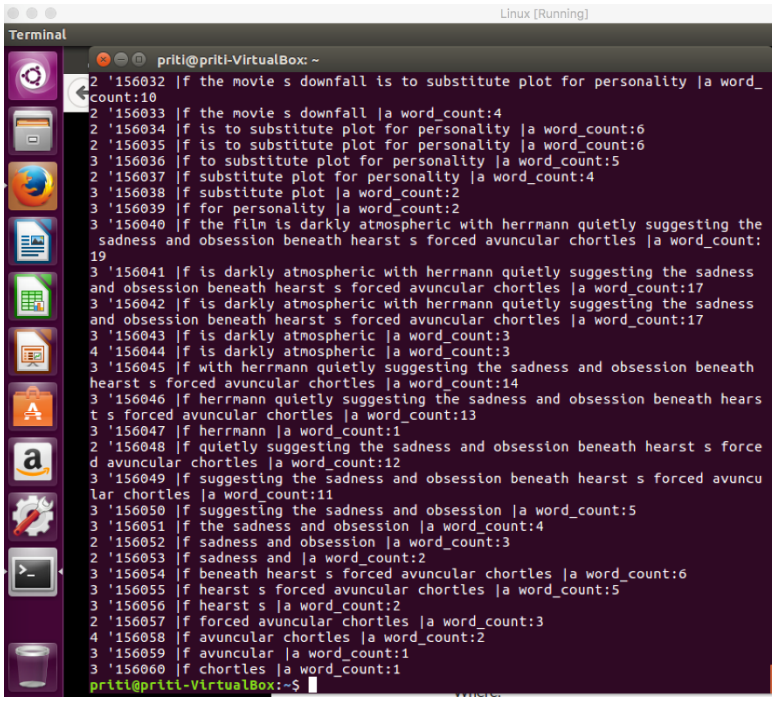
Test data.tsv:

```
PhraseId	SentenceId	Phrase
156061	8545	An intermittently pleasing but mostly routine effort .
156062	8545	An intermittently pleasing but mostly routine effort
156063	8545	An
156064	8545	intermittently pleasing but mostly routine effort
156065	8545	intermittently pleasing but mostly routine
156066	8545	intermittently pleasing but
156067	8545	intermittently pleasing
156068	8545	intermittently
156069	8545	pleasing
156070	8545	but
156071	8545	mostly routine
156072	8545	mostly
156073	8545	routine
156074	8545	effort
156075	8545	.
156076	8546	Kidman is really the only thing that 's worth watching in Birthday Girl , a film by the stage-trained Jez Butterworth -LRB- Mojo -RRB- that serves as yet another example of the sad decline of British comedies in the post-Full Monty world .
156077	8546	Kidman
156078	8546	is really the only thing that 's worth watching in Birthday Girl , a film by the stage-trained Jez Butterworth -LRB- Mojo -RRB- that serves as yet another example of the sad decline of British comedies in the post-Full Monty world .
156079	8546	is really the only thing that 's worth watching in Birthday Girl , a film by the stage-trained Jez Butterworth -LRB- Mojo -RRB- that serves as yet another example of the sad decline of British comedies in the post-Full Monty world
156080	8546	is really
156081	8546	is
156082	8546	really
156083	8546	the only thing that 's worth watching in Birthday Girl , a film by the stage-trained Jez Butterworth -LRB- Mojo -RRB- that serves as yet another example of the sad decline of British comedies in the post-Full Monty world
156084	8546	the only thing
156085	8546	the
156086	8546	only thing
156087	8546	only
156088	8546	thing
156089	8546	that 's worth watching in Birthday Girl , a film by the stage-trained Jez Butterworth -
```

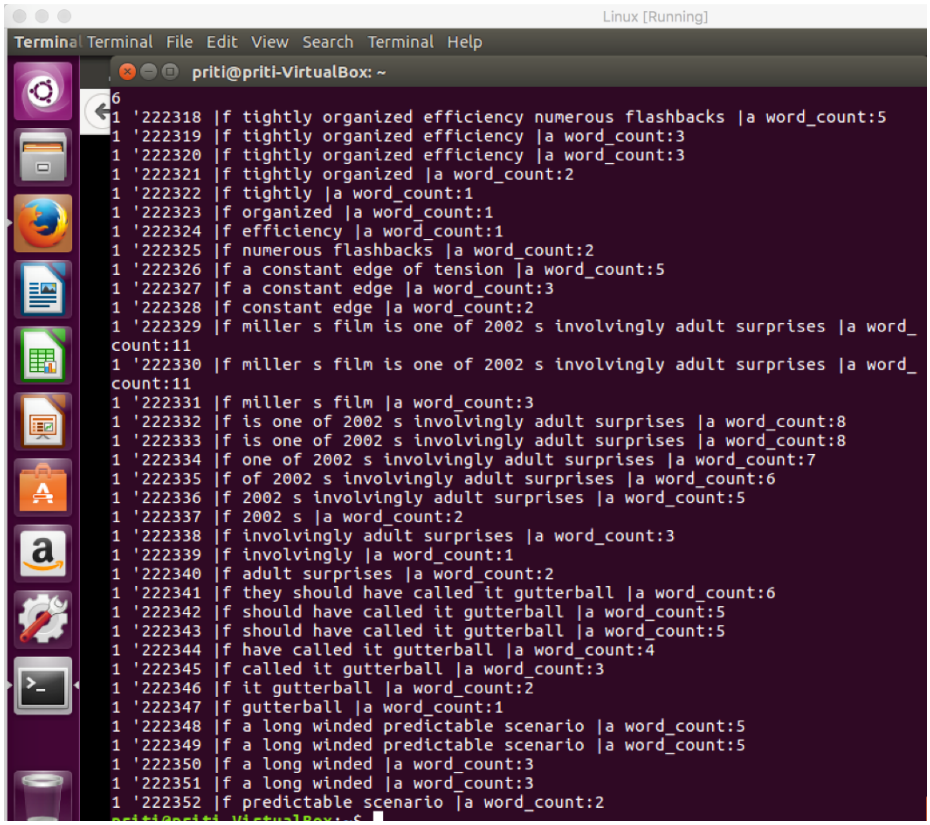
Train data.tsv

```
PhraseId	SentenceId	Phrase	Sentiment
1	1	A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which occasionally amuses but none of which amounts to much of a story .	1
2	1	A series of escapades demonstrating the adage that what is good for the goose	2
3	1	A series	2
4	1	A	2
5	1	series	2
6	1	of escapades demonstrating the adage that what is good for the goose	2
7	1	of	2
8	1	escapades demonstrating the adage that what is good for the goose	2
9	1	escapades	2
10	1	demonstrating the adage that what is good for the goose	2
11	1	demonstrating the adage	2
12	1	demonstrating	2
13	1	the adage	2
14	1	the	2
15	1	adage	2
16	1	that what is good for the goose	2
17	1	that	2
18	1	what is good for the goose	2
19	1	what	2
20	1	is good for the goose	2
21	1	is	2
22	1	good for the goose	3
23	1	good	3
24	1	for the goose	2
25	1	for	2
26	1	the goose	2
27	1	goose	2
28	1	is also good for the gander , some of which occasionally amuses but none of which amounts to much of a story .	2
29	1	is also good for the gander , some of which occasionally amuses but none of which amounts to much of a story	2
30	1	is also	2
31	1	also	2
32	1	good for the gander , some of which occasionally amuses but none of which amounts to much of a story	2
33	1	for the gander , some of which occasionally amuses but none of which amounts to much of a story	2
34	1	the gander , some of which occasionally amuses but none of which amounts to much of a story	1
35	1	the gander ,	2
36	1	the gander	2
```

Train.vw



Test.vw



create model:

```
priti@priti-VirtualBox:~$ vw rotten.train.vw -c -k --passes 300 --ngram 7 -b 24 --ect 5 -f rotten.model.vw
Generating 7-grams for all namespaces.
final_regressor = rotten.model.vw
Num weight bits = 24
learning rate = 0.5
initial_t = 0
power_t = 0.5
decay_learning_rate = 1
creating cache_file = rotten.train.vw.cache
Reading datafile = rotten.train.vw
num sources = 1
average  since         example        example  current  current  current
loss     last          counter         weight    label  predict features
1.000000 1.000000            1            1.0        2        1      226
1.000000 1.000000            2            2.0        3        2       79
0.500000 0.000000            4            4.0        3        3        3
0.250000 0.000000            8            8.0        3        3       58
0.125000 0.000000           16           16.0        3        3        3
0.156250 0.187500           32           32.0        3        3        5
```

```
average  since         example        example  current  current  current
loss     last          counter         weight    label  predict features
1.000000 1.000000            1            1.0        2        1      226
1.000000 1.000000            2            2.0        3        2       79
0.500000 0.000000            4            4.0        3        3        3
0.250000 0.000000            8            8.0        3        3       58
0.125000 0.000000           16           16.0        3        3        3
0.156250 0.187500           32           32.0        3        3        5
0.187500 0.218750           64           64.0        4        4        8
0.296875 0.406250          128          128.0        2        3        3
0.371094 0.445312          256          256.0        3        3        3
0.363281 0.355469          512          512.0        3        3       37
0.343750 0.324219         1024         1024.0        3        3        3
0.344238 0.344727         2048         2048.0        3        3        3
0.362305 0.380371         4096         4096.0        3        3        3
0.381714 0.401123         8192         8192.0        2        3       12
0.380371 0.379028        16384        16384.0        3        3        5
0.385010 0.389648        32768        32768.0        1        3      121
0.388138 0.391266        65536        65536.0        2        2       12
0.388260 0.388382       131072       131072.0        3        3       86
0.370927 0.370927       262144       262144.0        4        2       58 h
0.358585 0.346242       524288       524288.0        2        2       44 h
0.351203 0.343822      1048576      1048576.0        3        3       72 h

finished run
number of examples per pass = 140454
passes used = 11
weighted example sum = 1544994.000000
weighted label sum = 0.000000
average loss = 0.341856 h
total feature number = 52087112
priti@priti-VirtualBox:~$
```

Run vw in test mode and create predictions:

```
total feature number = 52087112
priti@priti-VirtualBox:~$ vw rotten.test.vw -t -i rotten.model.vw -p rotten.preds.txt
Generating 7-grams for all namespaces.
only testing
predictions = rotten.preds.txt
Num weight bits = 24
learning rate = 0.5
initial_t = 0
power_t = 0.5
using no cache
Reading datafile = rotten.test.vw
num sources = 1
average  since         example        example  current  current  current
loss     last          counter         weight    label  predict features
1.000000 1.000000            1            1.0        1        4       30
1.000000 1.000000            2            2.0        1        4       30
1.000000 1.000000            4            4.0        1        4       23
1.000000 1.000000            8            8.0        1        3        3
1.000000 1.000000           16           16.0        1        3      282
1.000000 1.000000           32           32.0        1        3      191
1.000000 1.000000           64           64.0        1        3        3
1.000000 1.000000          128          128.0        1        3       37
1.000000 1.000000          256          256.0        1        3       23
0.998047 0.996094          512          512.0        1        3        3
Amazon   .984375         1024         1024.0        1        3        3
0.993164 0.995117         2048         2048.0        1        4        3
0.989258 0.985352         4096         4096.0        1        3        3
0.990356 0.991455         8192         8192.0        1        3        3
0.987732 0.985107        16384        16384.0        1        3        3
0.986450 0.985168        32768        32768.0        1        3       17
0.987030 0.987610        65536        65536.0        1        3       17

finished run
number of examples per pass = 66292
passes used = 1
weighted example sum = 66292.000000
weighted label sum = 0.000000
average loss = 0.986831
total feature number = 2055004
priti@priti-VirtualBox:~$
```

rotten.preds.txt

```
priti@priti-VirtualBox:~$ head rotten.preds.txt
4 156061
4 156062
3 156063
4 156064
4 156065
3 156066
4 156067
3 156068
3 156069
3 156070
priti@priti-VirtualBox:~$
```

kaggle file format:

```
priti@priti-VirtualBox:~$ cat kaggleform.py
import csv
def to_kaggle(location_input_file, header="", location_output_file="kaggle.submission.csv"):
  print "\nReading:",location_input_file,"\nWriting:",location_output_file
  with open(location_input_file) as infile, open(location_output_file, "wb") as outfile:
    if len(header) > 0:
          outfile.write( header + "\n" )
    reader = csv.reader(infile, delimiter=" ")
    for row in reader:
      outfile.write( row[1] + "," + str(int(row[0][0])-1) + "\n" )

to_kaggle("rotten.preds.txt", "PhraseId,Sentiment")
```

kaggle format:

```
priti@priti-VirtualBox:~$ head kaggle.submission.csv
PhraseId,Sentiment
156061,3
156062,3
156063,2
156064,3
156065,3
156066,2
156067,3
156068,2
156069,2
```

R:

install.packages("devtools")
devtools::install_github("JohnLangford/vowpal_wabbit", subdir = "R/r.vw")

```r
library(r.vw)

## data
data("diamonds", package = "ggplot2")
dt = diamonds
dt$y = with(dt, ifelse(y < 5.71, 1, -1))

## separate train and validation data
ind_train = sample(1:nrow(dt), 40000)
dt_train = dt[ind_train,]
dt_val = dt[-ind_train,]


## first method: creating the vw data files before training
dt2vw(data = dt_train, fileName = "diamond_train.vw", target = "y")
dt2vw(data = dt_val, fileName = "diamond_val.vw", target = "y")

write.table(x = dt_val$y, file = "valid_labels.txt", row.names = F,
            col.names = F)

auc1 = vw(training_data = "diamond_train.vw", validation_data = "diamond_val.vw",
          validation_labels = "valid_labels.txt", use_perf = F)

## 2 method: use directly data.frames
auc2 = vw(training_data = dt_train, validation_data = dt_val,
          target = "y", use_perf = F)
```

dt = diamonds
> View(dt)

Filter

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39
11	0.30	Good	J	SI1	64.0	55.0	339	4.25	4.28	2.73
12	0.23	Ideal	J	VS1	62.8	56.0	340	3.93	3.90	2.46
13	0.22	Premium	F	SI1	60.4	61.0	342	3.88	3.84	2.33
14	0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
15	0.20	Premium	E	SI2	60.2	62.0	345	3.79	3.75	2.27
16	0.32	Premium	E	I1	60.9	58.0	345	4.38	4.42	2.68
17	0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68
18	0.30	Good	J	SI1	63.4	54.0	351	4.23	4.29	2.70
19	0.30	Good	J	SI1	63.8	56.0	351	4.23	4.26	2.71
20	0.30	Very Good	J	SI1	62.7	59.0	351	4.21	4.27	2.66
21	0.30	Good	I	SI2	63.3	56.0	351	4.26	4.30	2.71

dt$y = with(dt, ifelse(y < 5.71, 1, -1))

> View(dt)

Filter

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	1	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	1	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	1	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	1	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	1	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	1	2.48
7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	1	2.47
8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	1	2.53
9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	1	2.49
10	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	1	2.39
11	0.30	Good	J	SI1	64.0	55.0	339	4.25	1	2.73
12	0.23	Ideal	J	VS1	62.8	56.0	340	3.93	1	2.46
13	0.22	Premium	F	SI1	60.4	61.0	342	3.88	1	2.33
14	0.31	Ideal	J	SI2	62.2	54.0	344	4.35	1	2.71
15	0.20	Premium	E	SI2	60.2	62.0	345	3.79	1	2.27
16	0.32	Premium	E	I1	60.9	58.0	345	4.38	1	2.68
17	0.30	Ideal	I	SI2	62.0	54.0	348	4.31	1	2.68

dt2vw(data = dt_train, fileName = "diamond_train.vw", target = "y")

dt2vw(data = dt_val, fileName = "diamond_val.vw", target = "y")

```
pagarwal-retina-2016:Desktop pagarwal$ head diamond_train.vw
-1.000000 |A carat:1.200000 depth:59.800000 table:63.000000 price:11913.000000 x:6.820000 z:4.070000 cut_Very_Good color_F clarity_VVS1
1.000000 |A carat:0.350000 depth:61.300000 table:56.000000 price:522.000000 x:4.540000 z:2.790000 cut_Ideal color_G clarity_SI1
1.000000 |A carat:0.530000 depth:62.300000 table:55.000000 price:1812.000000 x:5.180000 z:3.240000 cut_Ideal color_F clarity_VS1
-1.000000 |A carat:1.530000 depth:60.800000 table:59.000000 price:11413.000000 x:7.410000 z:4.490000 cut_Premium color_H clarity_VS1
-1.000000 |A carat:1.010000 depth:66.900000 table:58.000000 price:2683.000000 x:6.130000 z:4.080000 cut_Fair color_J clarity_SI2
-1.000000 |A carat:1.000000 depth:59.200000 table:60.000000 price:7056.000000 x:6.530000 z:3.850000 cut_Premium color_F clarity_VS1
-1.000000 |A carat:1.310000 depth:60.800000 table:58.000000 price:6529.000000 x:7.130000 z:4.320000 cut_Premium color_H clarity_SI2
-1.000000 |A carat:1.000000 depth:68.300000 table:61.000000 price:3965.000000 x:6.020000 z:4.080000 cut_Fair color_G clarity_SI1
-1.000000 |A carat:1.280000 depth:63.000000 table:56.000000 price:6566.000000 x:6.900000 z:4.390000 cut_Very_Good color_F clarity_SI1
-1.000000 |A carat:1.000000 depth:63.900000 table:60.000000 price:4737.000000 x:6.220000 z:3.990000 cut_Good color_G clarity_SI1
pagarwal-retina-2016:Desktop pagarwal$
```

```
pagarwal-retina-2016:Desktop pagarwal$ head diamond_val.vw
1.000000 |A carat:0.290000 depth:62.400000 table:58.000000 price:334.000000 x:4.200000 z:2.630000 cut_Premium color_I clarity_VS2
1.000000 |A carat:0.300000 depth:62.000000 table:54.000000 price:348.000000 x:4.310000 z:2.680000 cut_Ideal color_I clarity_SI2
1.000000 |A carat:0.300000 depth:63.300000 table:56.000000 price:351.000000 x:4.260000 z:2.710000 cut_Good color_I clarity_SI2
1.000000 |A carat:0.230000 depth:63.800000 table:55.000000 price:352.000000 x:3.850000 z:2.480000 cut_Very_Good color_E clarity_VS2
1.000000 |A carat:0.310000 depth:59.400000 table:62.000000 price:353.000000 x:4.390000 z:2.620000 cut_Very_Good color_J clarity_SI1
1.000000 |A carat:0.310000 depth:58.100000 table:62.000000 price:353.000000 x:4.440000 z:2.590000 cut_Very_Good color_J clarity_SI1
1.000000 |A carat:0.300000 depth:62.200000 table:57.000000 price:357.000000 x:4.280000 z:2.670000 cut_Very_Good color_J clarity_VS2
1.000000 |A carat:0.230000 depth:60.900000 table:57.000000 price:357.000000 x:3.960000 z:2.420000 cut_Very_Good color_F clarity_VS1
1.000000 |A carat:0.230000 depth:60.000000 table:57.000000 price:402.000000 x:4.000000 z:2.410000 cut_Very_Good color_F clarity_VS1
1.000000 |A carat:0.230000 depth:59.800000 table:57.000000 price:402.000000 x:4.040000 z:2.420000 cut_Very_Good color_F clarity_VS1
```

write.table(x = dt_val$y, file = "valid_labels.txt", row.names = F,
col.names = F)

```
pagarwal-retina-2016:Desktop pagarwal$ head valid_labels.txt
1
1
1
1
1
1
1
1
1
1
```

auc1 = vw(training_data = "diamond_train.vw", validation_data = "diamond_val.vw",validation_labels = "valid_labels.txt", use_perf = F)

```
Model parameters
vw -d diamond_train.vw --loss_function logistic -f mdl.vw --learning_rate=0.5 --passes 1 -c -b 25
final_regressor = mdl.vw
Num weight bits = 25
learning rate = 0.5
initial_t = 0
power_t = 0.5
using cache_file = diamond_train.vw.cache
ignoring text input in favor of cache input
num sources = 1
average  since         example        example  current  current  current
loss     last          counter         weight    label  predict features
0.693147 0.693147           1            1.0  -1.0000   0.0000       10
0.953526 1.213904           2            2.0   1.0000  -0.8615       10
0.741037 0.528548           4            4.0   1.0000   0.6465       10
0.741710 0.742382           8            8.0  -1.0000   0.2557       10
0.738407 0.735104          16           16.0   1.0000  -0.6437       10
0.700023 0.661640          32           32.0  -1.0000  -0.0982       10
0.618492 0.536960          64           64.0   1.0000   0.2645       10
0.585926 0.553361         128          128.0  -1.0000  -1.1482       10
0.518052 0.450177         256          256.0   1.0000   0.1168       10
0.447227 0.376401         512          512.0   1.0000   1.7488       10
0.391185 0.335143        1024         1024.0   1.0000   1.4063       10
0.332577 0.273969        2048         2048.0   1.0000   1.4887       10
0.284353 0.236129        4096         4096.0   1.0000   1.9217       10
0.244951 0.205549        8192         8192.0   1.0000   1.1829       10
0.207620 0.170289       16384        16384.0  -1.0000  -7.1055       10
0.177196 0.146772       32768        32768.0   1.0000   3.4754       10
```

```
finished run
number of examples per pass = 40000
passes used = 1
weighted example sum = 40000.000000
weighted label sum = 34.000000
average loss = 0.169379
best constant = 0.001700
best constant's loss = 0.693147
total feature number = 399978
only testing
predictions = /var/folders/xt/hwwjj3g50rq1jvrjg97_gmww0000gn/T//RtmpFMfol5/preds.vw
Num weight bits = 25
learning rate = 0.5
initial_t = 0
power_t = 0.5
using no cache
Reading datafile = diamond_val.vw
num sources = 1
average  since         example        example  current  current  current
loss     last          counter         weight    label  predict features
9.783494 9.783494            1            1.0   1.0000   0.9841       10
11.795212 13.806931            2            2.0   1.0000   0.9911       10
7.852808 3.910404            4            4.0   1.0000   0.8990       10
8.221523 8.590238            8            8.0   1.0000   0.9885       10
8.656222 9.090920           16           16.0   1.0000   0.9854       10
6.801877 4.947532           32           32.0   1.0000   0.8270       10
3.826868 0.851858           64           64.0  -1.0000   0.2771       10
2.543511 1.260155          128          128.0  -1.0000   0.5427       10
1.777443 1.011376          256          256.0  -1.0000   0.4737       10
1.605330 1.433217          512          512.0  -1.0000   0.3110       10
1.681072 1.756814         1024         1024.0  -1.0000   0.1623       10
2.534038 3.387005         2048         2048.0  -1.0000   0.0068       10
5.664034 8.794029         4096         4096.0  -1.0000   0.0312       10
35.380604 65.097175         8192         8192.0   1.0000   0.9435       10

finished run
number of examples per pass = 13940
passes used = 1
weighted example sum = 13940.000000
weighted label sum = -200.000000
average loss = 22.315909
best constant = -0.014347
best constant's loss = 0.999794
total feature number = 139394


Call:
roc.default(response = labels, predictor = probs, auc = TRUE,     print.auc = TRUE, print.thres = TRUE)

Data: probs in 7070 controls (labels -1) < 6870 cases (labels 1).
Area under the curve: 0.997
Model Parameters
vw -d diamond_train.vw --loss_function logistic -f mdl.vw --learning_rate=0.5 --passes 1 -c -b 25[1] "AUC: 0.996978921535323"
```

```
$auc
[1] 0.9969789

$preds
    [1] 0.984138 0.991126 0.971597 0.899022 0.896377 0.982511 0.988508 0.988477 0.989711 0.849914 0.986347 0.986418 0.986638
   [14] 0.780184 0.993810 0.985383 0.989611 0.993696 0.993389 0.805706 0.993020 0.988043 0.914179 0.231057 0.155993 0.474843
   [27] 0.264816 0.444731 0.540982 0.473587 0.198308 0.827029 0.531690 0.655929 0.575906 0.117927 0.128676 0.407718 0.497276
   [40] 0.584677 0.811316 0.348083 0.354818 0.652554 0.323502 0.420799 0.539584 0.385025 0.531298 0.257436 0.504970 0.549275
   [53] 0.650703 0.513996 0.526799 0.501952 0.513873 0.639432 0.110398 0.855346 0.623935 0.110761 0.292750 0.277145 0.628594
   [66] 0.866411 0.266377 0.426761 0.034417 0.524010 0.888640 0.510727 0.578267 0.622317 0.589862 0.536093 0.643985 0.502572
   [79] 0.774405 0.898345 0.460963 0.202717 0.348222 0.200208 0.069846 0.345598 0.865271 0.290727 0.105461 0.160173 0.458004
   [92] 0.569827 0.477280 0.609871 0.562748 0.208940 0.618452 0.255695 0.501173 0.881693 0.554058 0.324455 0.545399 0.316500
  [105] 0.341601 0.369104 0.941278 0.939292 0.946090 0.980185 0.978773 0.953847 0.890590 0.534996 0.468597 0.333589 0.354644
  [118] 0.493399 0.732924 0.516396 0.075895 0.253647 0.611068 0.467018 0.479397 0.466999 0.753865 0.542720 0.504055 0.641359
  [131] 0.261074 0.594625 0.539844 0.119062 0.598589 0.352418 0.486763 0.586704 0.248022 0.867722 0.248951 0.113430 0.852846
  [144] 0.287848 0.498853 0.369738 0.508171 0.370459 0.135135 0.489157 0.379238 0.877002 0.849838 0.538060 0.632834 0.343888
  [157] 0.550106 0.555232 0.481853 0.507448 0.192939 0.187526 0.188342 0.263675 0.508833 0.562631 0.304671 0.616223 0.188194
  [170] 0.463547 0.209062 0.435123 0.638864 0.475009 0.516884 0.486824 0.837487 0.316392 0.557659 0.222630 0.383495 0.544784
  [183] 0.284735 0.263525 0.627942 0.157082 0.072050 0.925574 0.970298 0.976727 0.976585 0.976384 0.917626 0.913330 0.500301
  [196] 0.216946 0.292209 0.633889 0.471865 0.626782 0.521159 0.237549 0.057661 0.751418 0.378278 0.068179 0.169636 0.622153
```