Assignment 9

CS 532: Introduction to Web Science

Spring 2017

Grant Atkins

Finished on April 30, 2017

1

Question

1. Choose a blog or a newsfeed (or something similar with an Atom or RSS feed). Every student should do a unique feed, so please "claim" the feed on the class email list (first come, first served). It should be on a topic or topics of which you are qualified to provide classification training data. Find something with at least 100 entries (or items if RSS).

Create between four and eight different categories for the entries in the feed:

examples:

work, class, family, news, deals

liberal, conservative, moderate, libertarian

sports, local, financial, national, international, entertainment

metal, electronic, ambient, folk, hip-hop, pop

Download and process the pages of the feed as per the week 12 class slides.

Be sure to upload the raw data (Atom or RSS) to your github account.

Answer

For this problem I decided to use Realm's RSS feed, a news feed for mobile development [3]. I had 7 categories to classify this feed which were: iOS, Android, React Native, Realm News, Databases, Nodejs and Xamarin. These categories mostly describe the topics that were found in these news feeds. Realm News was used to describe topics that weren't particularly programming fields or platforms but just generic news topics. Realm's RSS feed originally had approximately 700 items, I decided to take the most recent 100 items with the full list shown in my github repository [1]. One thing I noticed before starting the subsequent problems was that a lot of the descriptions used common terms but in different context. For example, "Reactive programming" is not a synonym for "React Native" which was a programming language, this lead me to think that the classifier would later classify these items incorrectly.

To get Realm's RSS feed I used the methods created in **getFeed.py** shown in Listing 1. The manual classification of the 100 RSS feed items was created in a seperate text file called **classifiedFeeds.txt**. This was used later for speeding up running the python script. The 100 items used in this assignment are shown in Table 2 and Table 3 with their manual classification.

Category	**Count**
Nodejs	3
Android	19
Databases	5
Xamarin	5
React Native	4
iOS	41
Realm News	23

Table 1: Category and their count

```
1
2  import feedparser
3  import os.path
4  import requests
5  import html
6  import csv
7  import re
8
9
10 def saveFeed():
11     uri = "http://feeds.feedburner.com/realmio"
12     try:
13         filename = "data/feed.xml"
14         if not os.path.isfile(filename):
15             resp = requests.get(uri, stream=False,
                       allow_redirects=True, headers={
16                     'User-Agent': 'Mozilla/5.0'})
17
18             with open(filename, 'w') as f:
19                 f.write(resp.text)
20
21     except:
22         print("Could not get feed")
23         exit()
24
25
26 def remove_img_tags(data):
27     '''
28     Helper to remove img tags from description
29     '''
30     p = re.compile(r'<img.*?/>')
31     return p.sub('', data)
32
33
34 def remove_emojis(data):
35     '''
36     Helper to remove emojis from description
37     '''
38     emoji_pattern = re.compile("["
39                                u"\U0001F600-\U0001F64F"  #
                                       emoticons
40                                u"\U0001F300-\U0001F5FF"  #
                                       symbols & pictographs
41                                u"\U0001F680-\U0001F6FF"  #
                                       transport & map symbols
42                                u"\U0001F1E0-\U0001F1FF"  # flags
                                       (iOS)
43                                "]+", flags=re.UNICODE)
```

```
44      d = emoji_pattern.sub(r'', data)
45      return d
46
47
48  def select100(items):
49      filename = "data/feed100.txt"
50      if not os.path.isfile(filename):
51          with open(filename, 'w') as f, open("data/
                classifiedFeeds.txt", 'r') as o:
52              cat = {}
53              for j, item in enumerate(o):
54                  cat[j] = item
55
56              for i, entry in enumerate(items.entries):
57                  if i < 100:
58                      print(entry["title"])
59                      # remove html encodings like &amp;
60                      t = html.unescape(entry["title"])
61                      d = html.unescape(entry["description"])
62                      d = remove_img_tags(d)
63                      d = remove_emojis(d)
64                      for key, value in cat.items():
65                          if i == key:
66                              f.write(t + "|" + d + "|" + item)
67                  else:
68                      break
69
70
71  def recreate100():
72      with open("data/feed100.txt", 'r') as f, open("data/
            classifiedFeeds.txt", 'w') as out:
73          reader = csv.reader(f, delimiter='|')
74          for i in reader:
75              categ = i[1]
76              out.write(categ + '\n')
77
78
79  def createTabular():
80      with open("data/feed100.txt", 'r') as f, open("../docs/
            tabular.tex",
81                                                    'w') as t:
82          reader = csv.reader(f, delimiter="|")
83          for i in reader:
84              title = i[0].replace('&', '\&')
85              newsType = i[1]
86              outStr = title + " & " + newsType + " \\\\ \n"
87              outStr += "\hline\n"
88              t.write(outStr)
89
```

```
90
91 if __name__ == "__main__":
92     saveFeed()
93     items = feedparser.parse(r"data/feed.xml")
94     select100(items)
95     # createTabular()
96     # recreate100()
```

Listing 1: Python script to download Realm RSS Feed

Item Title	Actual Category
Everyday Reactive	React Native
Providing Better Feedback in Realtime Object Detection Apps	iOS
Fragments: The Solution to (and Cause of) All of Android's Problems	Android
Realm ObjC & Swift 2.6: Async Open Realm, Admin Flag, Compact On Launch & Bug Fixes!	iOS
Building a Gantt Chart from Github Issues: With Near Caching Using Realm	Realm News
Building Your Own Tools	React Native
Server-Side Swift Live Coding	iOS
Legal Risks and Rights for Developers	Realm News
The Post-MVC Age	React Native
Making Mock Objects More Useful	iOS
Exploring New Android Layouts	Android
Crafting Collaborative Apps with Realm	Realm News
Visual Studio Code: Shipping One of the Largest Microsoft JavaScript Apps	Xamarin
Swift on Android: The Future of Cross-Platform Programming or White Whale?	Android
Realm Java 3.1: Object Notifications, Backup Recovery and Reverse Relationships	Android
Reverse Engineering Is Not Just for Hackers	Android
Git at Scale: Managing Swift/Obj-C Code & Coders	iOS
Realm World Tour: Thats a Wrap Time for Round Two!	Realm News
Swift's Pointy Bits: Unsafe Swift & Pointer Types	iOS
Tutorial: Build iOS App from Scratch	iOS
Realm ObjC & Swift 2.5: Query Improvements, Swift 3.1 Binaries & Bug Fixes!	iOS
Data Binding in the Real World	Android
Taming Node_Modules at Facebook	Nodejs
The Safety of Unsafe Swift	iOS
Realm Browser Tutorial	Realm News
Scaling Open Source Communities	Realm News
Evolution in Action: Software Architecture as Systems Dissolve	Realm News
Bring Your Own Authentication: Connect Your Users to the Realm Mobile Platform	Realm News
No More Typos: Foolproof Notifications in Swift	iOS
Getting Down to Business With Firebase Monitoring Tools	Databases
Making PostgreSQL Realtime	Databases
Everything You Ever Wanted To Know About Sequence and Collection	iOS
Building a Blog with Realm Node.js and Express	Nodejs
Apple and VR	iOS
Espresso: Beyond the Basics	Android
Break the Monolith with (B)Viper Modules	iOS
UI and Snapshot Testing	iOS
Scaling Your App for Rapid Growth by using Testing, Deploying and Monitoring	Xamarin
Realm Everywhere, with JavaScript: Announcing Universal Node.js Support	Nodejs
Revisiting Types in Kotlin	Android
Realm Primary Keys Tutorial	Databases
Integrating Azure Authentication with Realm	Xamarin
Hacking SiriKit	iOS
Test Driven Development (TDD) for the Masses	iOS
VoiceOver is Awesome	iOS
Interacting with Your App Through the Command Line	Android
Realm Java 3.0: Collection Notifications, Snapshots and Sorting Across Relationships	Android
Swift at Scale	iOS
Playgrounds: teach nerdy stuff in a fun and efficient way!	iOS
Sell Out and Save the World!	Realm News
Testing Functional Reactive Programming Code	iOS
Functional on Android: Lambdas, Rx, and Streams in Your App	Android
Pushing the Boundaries of Swift to the Server	iOS
Stylish Developers Guide to Unit Testing in Swift	iOS
The History of Mac and iOS: Squirrels, Disco, and Nate Eror	iOS
Selling Your Weird Mouth Noises	Realm News
Acceptance Testing	iOS
RxJava for the Rest of Us	Android
Realm + Microsoft: Xamarin, Azure, and Windows Desktop	Xamarin
Data Consistency in an Unpredictable World	iOS
Network Testing	iOS
Be the Quality You Want to See in Your App [Swift edition]	iOS

Table 2: Realm news item manual classifications 1-62

Item Title	Actual Category
The 2016 Android Developer Toolbox	Android
Realm Cocoa Tutorial: Encryption with Realm	iOS
MVVM with Coordinators and RxSwift	iOS
Visualize, Document, and Explore Your Software Architecture	Realm News
How Indies Can Still Impact the Future of iTunes	Realm News
Realm React Native 1.0: Powerful Object Database Meets the Realm Mobile Platform	React Native
Creating the Future	iOS
Compile Time Errors Are Good	iOS
Better Android Development with Kotlin & Gradle	Android
Realm: How I Learned to Love Databases Again	Databases
Operators and Strong Opinions	iOS
Realm ObjC & Swift 2.4: Object Notifications!	iOS
A Startups Secret Weapon: The Product Engineer	Realm News
Testing an Untested App	iOS
Writing Software to Make a Difference	Realm News
Radical Hospitality - One Shower at a Time	Realm News
MVC vs. MVP vs. MVVM on Android	Android
Bringing the Platform Experience to You: Announcing the Realm World Tour	Realm News
The Objective-C Runtime & Swift Dynamism	iOS
Reactive Apps: How to Build More Engaging Mobile Experiences	Realm News
Mastering Realm Notifications	iOS
Event Handling in the Realm Object Server	Realm News
A Designers Response to Silicon Valley	Realm News
Ready for Realtime and Scale: Announcing Realm Mobile Platform 1.0	Realm News
Eventually Consistent: How to Make a Mobile-First Distributed System	Realm News
Realm ObjC & Swift 2.3: Sync Progress Notifications, Improved Sharing & Backup Recovery!	iOS
Realm Java 2.3: Improved Sharing, Backup Recovery and Wildcard Queries	Android
Smoke & Mirrors: The Magic Behind Wonderful UI in Android	Android
JP Simard on Realm & Open Source on the Consult Podcast	Realm News
Testing in Swift: Protocols & View Models	iOS
Property-Based Testing with SwiftCheck	iOS
Realm Objective-C & Swift 2.2: Objects across threads, sort over relationships & more!	iOS
Contextual Communication in a Connected World	Realm News
Introduction to Xamarin Forms Custom Renderers	Xamarin
Modern Android: Ditching Activities and Fragments	Android
Safe vs Deep Integration of Realm	Android
Watch Your Language!: The Road to Cleaner Code with SwiftLint	iOS
Introducing the Realm Mobile Platform	Databases

Table 3: Realm news item manual classifications 63-100 cont.

2

Question

```
2.  Train the Fisher classifier on the first 50 entries (the "training
set"), then use the classifier to guess the classification of the
next 50 entries (the "test set").

Assess the performance of your classifier in each of your categories
by computing precision, recall, and F-measure.  Use the "macro-averaged"
label based method, as per:

http://stats.stackexchange.com/questions/21551/
how-to-compute-precision-recall-for-multiclass-multilabel-classification

For example, if you have 5 categories (e.g., 80s, metal,
alternative, electronic, cover), you will compute
precision, recall, and F-measure for each category,
and then compute the average across the 5 categories.
```

Answer

To solve this I used the python code provided from the Programming Collective Intelligence book described in the files **docclass.py** and **feedfilter.py**. I had to slightly modify docclass's first dependency to compile in python 3. I altered feedfilter's code heavily in the *read* method, creating new dictionaries for each of the Fisher classifier's guesses and actual answers as shown in Listing 2. Feedfilter extracted the title and description from each item in the RSS feed and would concatenate the two which would then be used for classifying and training, if the iteration count was lower than the desired train count which was defined as *trainCount*. Realm's RSS feed also had some items with quotation marks, emoji icons and img html tags. To compensate for this, I created two methods *remove_emojis* and *remove_img_tags* which removed these unwanted characters before guessing or training the classifier.

Combining the docclass and feedfilter scripts, I created a script called **classify.py** as shown in Listing 3 where I called the respective methods the helper classes and created a new method called *calcMeasurements* which takes the dictionaries created in feedfilter to calculate the precision, recall and F-measure.

When performing these calculations for each category, I iterated through each of the 100 items checking for False Positives, False Negatives and True Positives. These values were used for calculating the precision, recall and F-measure. Some of these values ended up being 0 which affected other calculations.

```
1  import feedparser
2  import re
3  import html
4
5  # Takes a filename of URL of a blog feed and classifies the
       entries
6
7
8  def read(feed, classifier, trainCount):
9      # Get feed entries and loop over them
10     f = feedparser.parse(feed)
11     # open known classified feeds
12     cat = {}
13     with open("data/classifiedFeeds.txt", "r") as cfeeds:
14         for j, item in enumerate(cfeeds):
15             cat[j] = item
16
17     # store results for probability later
18     results = []
19     for i, entry in enumerate(f.entries):
20         if(i == 100):
21             break
22         print()
23         print('-----')
24         # Print the contents of the entry
25         t = html.unescape(entry["title"])
26         t = t.replace('"', '')
27         d = html.unescape(entry["description"])
28         d = remove_img_tags(d)
29         d = remove_emojis(d)
30         d = d.replace('"', '')
31         print('Title:      ' + t)
32         print()
33         print(d)
34
35         entryDict = {}
36         entryDict['Title'] = t
37
38         # Combine all the text to create one item for the
               classifier
39         fulltext = '%s\n%s' % (t, d)
40
```

```
41          # Print the best guess at the current category
42          guess = str(classifier.classify(fulltext))
43          entryDict['Guess'] = guess
44          print('Guess: ' + guess)
45          cl = ""
46          for key, val in cat.items():
47              if(i == key):
48                  cl = val
49                  break
50
51          # remove newline
52          cl = cl.rstrip()
53          print('Actual Category: ' + cl)
54          entryDict['Actual'] = cl
55          results.append(entryDict)
56
57          if i <= trainCount:
58              # train on already known category
59              classifier.train(fulltext, cl)
60
61      return results
62
63
64  def remove_img_tags(data):
65      '''
66      Helper to remove img tags from description
67      '''
68      p = re.compile(r'<img.*?/>')
69      return p.sub('', data)
70
71
72  def remove_emojis(data):
73      '''
74      Helper to remove emojis from description
75      '''
76      emoji_pattern = re.compile("["
77                                 u"\U0001F600-\U0001F64F"  #
                                       emoticons
78                                 u"\U0001F300-\U0001F5FF"  #
                                       symbols & pictographs
79                                 u"\U0001F680-\U0001F6FF"  #
                                       transport & map symbols
80                                 u"\U0001F1E0-\U0001F1FF"  # flags
                                       (iOS)
81                                 "]+", flags=re.UNICODE)
82      d = emoji_pattern.sub(r'', data)
83      return d
84
85
```

```
86  def entryfeatures(entry):
87      splitter = re.compile('\\W*')
88      f = {}
89
90      # Extract the title words and annotate
91      titlewords = [s.lower() for s in splitter.split(entry['title
            '])
92                    if len(s) > 2 and len(s) < 20]
93      for w in titlewords:
94          f['Title:' + w] = 1
95
96      # Extract the summary words
97      summarywords = [s.lower() for s in splitter.split(entry['
            summary'])
98                      if len(s) > 2 and len(s) < 20]
99
100     # Count uppercase words
101     uc = 0
102     for i in range(len(summarywords)):
103         w = summarywords[i]
104         f[w] = 1
105         if w.isupper():
106             uc += 1
107
108         # Get word pairs in summary as features
109         if i < len(summarywords) - 1:
110             twowords = ' '.join(summarywords[i:i + 1])
111             f[twowords] = 1
112
113     # Keep creator and publisher whole
114     f['Publisher:' + entry['publisher']] = 1
115
116     # UPPERCASE is a virtual word flagging too much shouting
117     if float(uc) / len(summarywords) > 0.3:
118         f['UPPERCASE'] = 1
119
120     return f
```

Listing 2: Python script to process train and Classify RSS feed items

```
1
2   import docclass
3   import feedfilter
4   import os
5
6
7   def findProb(data, cl):
8       '''
9       Use predefined fisher methods for these probabilities
```

```
10      '''
11      for item in data:
12          item['Cprob'] = cl.cprob(item['Title'], item["Actual"])
13          item['fisherprob'] = cl.fisherprob(item['Title'], item["
                Actual"])
14          print(item)
15      return data
16
17
18  def calcMeasurements(cats, data, trainCount):
19      '''
20      Method to calculate fmeasure, precision, recall
21      TP = True Positive
22      FP = False Positive
23      FN = False Negative
24      '''
25      catDict = {}
26      for cat in cats:
27          TP, FP, FN = 0.0, 0.0, 0.0
28          for i, item in enumerate(data):
29              if item['Actual'] != cat:
30                  continue
31              if not item['Guess']:
32                  FN += 1.0
33              elif(item['Actual'] == item['Guess']):
34                  TP += 1.0
35              elif(item['Actual'] != item['Guess']):
36                  FP += 1.0
37              print(item['Title'], item['Guess'], item['Actual'],
                    "asdasd")
38
39          prec = TP / (TP + FP)
40          recall = 0.0
41          if (TP + FN) != 0.0:
42              recall = TP / (TP + FN)
43          f1 = 0.0
44          if (prec + recall) != 0.0:
45              f1 = 2 * (prec * recall) / (prec + recall)
46          print(cat)
47          print(prec)
48          print(recall)
49          print(f1)
50          catDict[cat] = {'prec': prec, 'recall': recall, 'f1': f1
                }
51
52      return catDict
53
54
55  def printTabular(cats):
```

```
56      outStr = "\hline\n"
57      print(cats)
58      for key, val in cats.items():
59          outStr += key + " & " + str(val['prec']) + " & " + \
60              str(val['recall']) + " & " + str(val['f1']) + " \\\\
                \n"
61
62      return outStr
63
64
65  def tabularPredictions(data, trainCount):
66      outStr = "\hline\n"
67      for i, item in enumerate(data):
68          if(i == trainCount):
69              outStr += "% TEST SET STARTS HERE\n"
70          outStr += item["Title"] + " & " + item["Actual"] + \
71              " & " + item["Guess"] + " \\\\ \n"
72
73      return outStr
74
75
76  def resultPred(data):
77      '''
78      Get set of categories and list of predictions
79      '''
80      cats = set()
81      predicted = []
82      for i in data:
83          cats.add(i["Actual"])
84          predicted.append(i["Guess"])
85      return cats, predicted
86
87
88  def removeDbs():
89      '''
90      Clean databases for each run
91      '''
92      db1 = "data/first90-trained.db"
93      db2 = "data/first50-trained.db"
94      if(os.path.isfile(db1)):
95          os.remove(db1)
96      if(os.path.isfile(db2)):
97          os.remove(db2)
98
99
100 if __name__ == "__main__":
101     removeDbs()
102     trainCount = 90
103     cl = docclass.fisherclassifier(docclass.getwords)
```

```
104     cl.setdb("data/first" + str(trainCount) + "-trained.db")
105     data = feedfilter.read("data/feed.xml", cl, trainCount)
106     data = findProb(data, cl)
107     cats, predicted = resultPred(data)
108     catDict = calcMeasurements(cats, data, trainCount)
109     # print to files. should only be 1 time run
110     # with open("../docs/" + str(trainCount) + "measureTabular.
            tex", 'w') as f:
111     #     outStr = printTabular(catDict)
112     #     f.write(outStr)
113     # with open("../docs/" + str(trainCount) + "trained.tex", 'w
            ') as f:
114     #     outStr = tabularPredictions(data, trainCount)
115     #     f.write(outStr)
```

Listing 3: Python script to calculate precision, recall and F-measure of classifier items

Item Title	Actual Category	Predicted Category
Testing Functional Reactive Programming Code	iOS	React Native
Functional on Android: Lambdas, Rx, and Streams in Your App	Android	Android
Pushing the Boundaries of Swift to the Server	iOS	iOS
Stylish Developers Guide to Unit Testing in Swift	iOS	iOS
The History of Mac and iOS: Squirrels, Disco, and Nate Eror	iOS	iOS
Selling Your Weird Mouth Noises	Realm News	iOS
Acceptance Testing	iOS	Realm News
RxJava for the Rest of Us	Android	Android
Realm + Microsoft: Xamarin, Azure, and Windows Desktop	Xamarin	Nodejs
Data Consistency in an Unpredictable World	iOS	Realm News
Network Testing	iOS	iOS
Be the Quality You Want to See in Your App [Swift edition]	iOS	Databases
The 2016 Android Developer Toolbox	Android	Android
Realm Cocoa Tutorial: Encryption with Realm	iOS	Databases
MVVM with Coordinators and RxSwift	iOS	Android
Visualize, Document, and Explore Your Software Architecture	Realm News	iOS
How Indies Can Still Impact the Future of iTunes	Realm News	React Native
Realm React Native 1.0: Powerful Object Database Meets the Realm Mobile Platform	React Native	Realm News
Creating the Future	iOS	Realm News
Compile Time Errors Are Good	iOS	iOS
Better Android Development with Kotlin & Gradle	Android	Android
Realm: How I Learned to Love Databases Again	Databases	Realm News
Operators and Strong Opinions	iOS	React Native
Realm ObjC & Swift 2.4: Object Notifications!	iOS	iOS
A Startup?s Secret Weapon: The Product Engineer	Realm News	iOS
Testing an Untested App	iOS	iOS
Writing Software to Make a Difference	Realm News	iOS
Radical Hospitality - One Shower at a Time	Realm News	iOS
MVC vs. MVP vs. MVVM on Android	Android	iOS
Bringing the Platform Experience to You: Announcing the Realm World Tour	Realm News	Realm News
The Objective-C Runtime & Swift Dynamism	iOS	iOS
Reactive Apps: How to Build More Engaging Mobile Experiences	Realm News	React Native
Mastering Realm Notifications	iOS	Nodejs
Event Handling in the Realm Object Server	Realm News	Realm News
A Designer?s Response to Silicon Valley	Realm News	React Native
Ready for Realtime and Scale: Announcing Realm Mobile Platform 1.0	Realm News	Nodejs
Eventually Consistent: How to Make a Mobile-First Distributed System	Realm News	Realm News
Realm ObjC & Swift 2.3: Sync Progress Notifications, Improved Sharing & Backup Recovery!	iOS	iOS
Realm Java 2.3: Improved Sharing, Backup Recovery and Wildcard Queries	Android	Android
Smoke & Mirrors: The Magic Behind Wonderful UI in Android	Android	Android
JP Simard on Realm & Open Source on the Consult Podcast	Realm News	Realm News
Testing in Swift: Protocols & View Models	iOS	Xamarin
Property-Based Testing with SwiftCheck	iOS	iOS
Realm Objective-C & Swift 2.2: Objects across threads, sort over relationships & more!	iOS	iOS
Contextual Communication in a Connected World	Realm News	Realm News
Introduction to Xamarin Forms Custom Renderers	Xamarin	Android
Modern Android: Ditching Activities and Fragments	Android	Android
Safe vs Deep Integration of Realm	Android	Android
Watch Your Language!: The Road to Cleaner Code with SwiftLint	iOS	Android
Introducing the Realm Mobile Platform	Databases	Xamarin

Table 4: Items 50-100 predicted by the Fisher classifier

Item Title	Precision	Recall	F-measure
Nodejs	0.333333	1.0	0.5
Android	0.526315	1.0	0.689655
Realm News	0.260869	1.0	0.413793
React Native	0.25	1.0	0.4
Databases	0.0	0.0	0.0
Xamarin	0.0	0.0	0.0
iOS	0.487804	1.0	0.655737
Averages	0.265474	0.714286	0.379884

Table 5: Precision, recall and F-measure calculations for the Fisher classifier

3

Question

```
3.  Repeat question #2, but use the first 90 entries to train your
classifier and the last 10 entries for testing.
```

Answer

To solve this question I used the same code from question 2 shown in Listing 3. I simply changed the value of *trainCount* to 90 and it worked the same way. The news items selected are shown in Table 6. The calculations for this problem are shown in Table 7.

Whats noticeable in this is that the Databases category was eventually predicted correctly, however the Xamarin category was still never predicted correctly. The Xamarin and Database categories had the same number of items, 5, however this probably means it just couldn't make use of the context of the description and title.

Item Title	Actual Category	Predicted Category
JP Simard on Realm & Open Source on the Consult Podcast	Realm News	Realm News
Testing in Swift: Protocols & View Models	iOS	iOS
Property-Based Testing with SwiftCheck	iOS	iOS
Realm Objective-C & Swift 2.2: Objects across threads, sort over relationships & more!	iOS	iOS
Contextual Communication in a Connected World	Realm News	iOS
Introduction to Xamarin Forms Custom Renderers	Xamarin	Android
Modern Android: Ditching Activities and Fragments	Android	Android
Safe vs Deep Integration of Realm	Android	Android
Watch Your Language!: The Road to Cleaner Code with SwiftLint	iOS	Android
Introducing the Realm Mobile Platform	Databases	Databases

Table 6: Items 90-100 predicted by the Fisher classifier

Item Title	Precision	Recall	F-measure
Nodejs	0.333333	1.0	0.5
Android	0.473684	1.0	0.642857
Databases	0.2	1.0	0.333333
Xamarin	0.0	0.0	0.0
React Native	0.25	1.0	0.4
iOS	0.536585	1.0	0.698412
Realm News	0.260869	1.0	0.413793
Averages	0.293496	0.857143	0.426914

Table 7: Precision, recall and F-measure calculations for the Fisher classifier

4

Question

```
4.  Rerun question 3, but with "10-fold cross validation".  What
was the change, if any, in precision and recall (and thus F-Measure)?
```

Answer

NOT ATTEMPTED

References

[1] Atkins, Grant. "CS532 Assignment 9 Repository" Github. N.p., 23 March 2017. Web. 23 March 2017.https://github.com/grantat/cs532-s17/tree/master/assignments/A9.

[2] Segaran, Toby. "Programming Collective Intelligence". O' Reilly, 2007. Web. 6 April 2017. http://shop.oreilly.com/product/9780596529321.do.

[3] "Realm now has an RSS feed". Realm, 24 Oct 2014. Web. 30 April 2017. https://news.realm.io/news/realm-rss-feed/.