



Module 12: Opinion Mining



What is Opinion Mining?

- OM is a recent discipline that studies the extraction of opinions using IR, AI and/or NLP techniques.
- More informally, it's about extracting the opinions or sentiments given in a piece of text
- Also referred to as Sentiment Analysis
- Web 2.0 nowadays provides a great medium for people to share things.
- This provides a great source of unstructured information (especially opinions) that may be useful to others (e.g. companies and their rivals, other consumers...)

Opinion Mining is Big Business

- Someone who wants to buy a camera
 - Looks for comments and reviews
- Someone who just bought a camera
 - Comments on it
 - Writes about their experience
- Camera Manufacturer
 - Gets feedback from customers
 - Improve their products
 - Adjust Marketing Strategies



Related (sub)topics: general

- **Opinion extraction:** extract the piece of text which represents the opinion
 - I just bought a new camera yesterday. It was a bit expensive, but the battery life is very good.
- **Sentiment classification/orientation:** extract the polarity of the opinion (e.g. positive, negative, neutral, or classify on a numerical scale)
 - negative: expensive
 - positive: good battery life
- **Opinion summarisation:** summarise the overall opinion about something
 - price:negative, battery life: positive --> overall 7/10

Feature-opinion association

- **Feature-opinion association:** given a text with target features and opinions extracted, decide which opinions comment on which features.
 - “The battery life is good but not so keen on the picture quality”
- **Target identification:** which thing is the opinion referring to?
- **Source identification:** who is holding the opinion?
- There may be attachment and co-reference issues
 - “The camera comes with a free case but I don't like the colour much.”
 - Does this refer to the colour of the case or the camera?

Spam opinion detection

- **Spam opinion detection:** detect whether opinions are written by spammers
- Could involve language analysis, frequency analysis,...
- Could be either positive or negative opinions
- Generally, negative opinions are more damaging than positive ones
- Spam opinions are typically posted by the same person on multiple sites and/or threads with identical or very similar wording
- The Chinese government paid people to post spam opinions supporting them: typically these are easily detectable due to their similarity and frequency
- Look for **outlier** reviews (which go against the trend)

Deep sentiment analysis

- The hardest thing about getting sentiment analysis right is uncovering exactly what is being meant
- Difference between a customer saying they merely like a brand and saying that they love it.
- Sentiment has many rich and nuanced dimensions that need to be teased apart to make it insightful.
- “An old lady told me that warm Dr. Pepper is delicious”
 - Is it only nice when warm?
 - Does the author share the opinion of the old lady?
 - Could this be a new insight for the manufacturers/advertisers?
- “I loved Dr. Pepper because of Dude.” (Dude was a character in the ad)
 - Does the author still like Dr Pepper?
 - Should the manufacturers consider bringing back the commercial?
- Classification of sentiment according to functional, insightful, emotional etc.

The Ultimate Question

- The book "The Ultimate Question" recently ranked #1 on the Wall Street Journal's Business Best-Sellers List and #1 on USA TODAY's Money Best-Sellers List.
- It's all about whether a consumer likes a brand enough to recommend it.
- This apparently is the key to a company's performance.
- General sentiment detection isn't precise enough to answer this kind of question, because all kinds of "like" are treated equally
- Growing need for sentiment analysis that can get to very fine levels of detail, while keeping up with the enormous (and constantly increasing) volume of social media.

All opinions are not equal

- Opinion Mining needs to take into account how much influence any single opinion is worth
- This could depend on a variety of factors, such as how much trust we have in a person's opinion, and even what sort of person they are
- Need to account for:
 - experts vs non-experts
 - spammers
 - frequent vs infrequent posters
 - “experts” in one area may not be expert in another
 - how frequently do other people agree?

Trust Recommenders

- Two types of trust:
 - relationship (local) trust
 - reputation (global) trust.
- **Relationship trust:** if you and I both rate the same things, and our opinions on them match closely, we have high relationship trust. This can be extended to a social networking group --> web of trust.
- **Reputation trust:** if you've recommended the same thing as other people, and usually your recommendation is close to what the majority of people think, then you're considered to be more of an expert and have high reputation trust.
- We can extend relationship trust to form clusters of interests and likes/dislikes
- We can narrow reputation trust to opinions about similar topics

Rule-based Opinion Mining from Political Tweets

Processing political tweets

- Application to associate people with their political leanings, based on pre-election tweets
- First stage is to find triple <Person, Opinion, Political Party>
 - e.g. John Smith is pro_Labour
- Usually, we will only get a single sentiment per tweet
- Later, we can collect all mentions of “John Smith” that refer to the same person, and collate the information
 - For example, John may be equally in favour of several different parties, not just Labour, but hates the Conservatives above all else

Creating a corpus

- First step is to create a corpus of tweets
- Created an API to suck up all the tweets over the pre-election period according to various criteria (e.g. use of certain hash tags, mention of various political parties etc.)
- Rather than collect just the tweets, we collect them in json format and then convert these to xml
- This means that we have lots of additional twitter metadata, such as the date and time of the tweet, the number of followers of the person tweeting, the location and other information about the person tweeting, and so on
- This information is useful for disambiguation and for collating the information later

Tweets with metadata

```
Thu Mar 25 20:06:32 +0000 2010 false 11050953883 <a
href="http://www.trinketsoftware.com/Twikini" rel="nofollow">Twikini</a>Had
pleasure of formally proposing Stuart King as Labour Candidate for Putney.
Yes he can..... false false Fri Jan 23 15:21:58 +0000 2009 Member of
Parliament for Tooting 0 4224 1590 false 19397942 en London, UK Sadiq
Khan MP f0feff
http://a3.twimg.com/profile_background_images/4356861/twitter.jpg false
http://a1.twimg.com/profile_images/427349972/playgroundcropped_normal.JPG
G 0084B4 BDDCAD DDFCC 333333 false SadiqKhan 1390 London
http://www.sadiqkhan.org.uk 0 false
```

Original markups set

- in_reply_to_user_id
- lang
- location
- name
- notifications
- o
- place
- profile_background_color
- profile_background_image_url
- profile_background_tile
- profile_image_url
- profile_link_color
- profile_sidebar_border_color
- profile_sidebar_fill_color
- profile_text_color
- protected
- screen_name
- source
- statuses_count
- text
- time_zone



Metadata

```

Thu Mar 25 20:06:32 +0000 2010 false 11050953883 <a
href="http://www.trinketsoftware.com/Twikini" rel="nofollow">Twikini</a>Had
pleasure of formally proposing Stuart King as Labour Candidate for Putney.
Yes he can..... false false Fri Jan 23 15:21:58 +0000 2009 Member of
Parliament for Tooting 0 4224 1590 false 19397942 en London, UK Sadiq
Khan MP f0feff
http://a3.twimg.com/profile_background_images/4356861/twitter.jpg false
http://a1.twimg.com/profile_images/427349972/playgroundcropped_normal.JP
G 0084B4 BDDCAD DDFFCC 333333 false SadiqKhan 1390 London
http://www.sadiqkhan.org.uk 0 false

```

Date

Tweet

Number of friends

Location

Profile info

Name

Gazetteers

- We create an instance of a flexible gazetteer to match certain useful keywords, in various morphological forms:
 - political parties, e.g. “Conservative”, “LibDem”
 - concepts about winning election, e.g. “win”, “landslide”
 - words for politicians, e.g. “candidate”, “MP”
 - words for voting and supporting a party/ person, e.g. “vote”
 - words indicating negation, e.g. “not”, “never”
- We create another gazetteer containing affect/emotion words from WordNet.
 - these have a feature denoting part of speech (category)
 - Keeping category information may be important, so we don't want a flexible gazetteer here

Grammar rules: creating temporary annotations

- Identify questions or doubtful statements as opposed to "factual" statements in tweets.
- Initially, we just look for question marks
 - *"Wont Unite's victory be beneficial to Labour?"*
- Create temporary Affect annotations if an "affect" Lookup is found and if the category matches the POS tag on the Token (this ensures disambiguation of the different possible categories)
 - *"Just watched video about awful days of Tory rule" vs "Ah good, the entertainment is here."*
 - *"People like her should be shot." vs "People like her."*

Question grammar

Phase: Preprocess

Input: Token

Options: control = appelt

Rule: Question

```
(  
  {Token.string == "?"}  
) :tag  
-->  
:tag.Question = {rule = "Question"}
```

Affect grammar

Phase: Affect

Input: AffectLookup Token

Options: control = appelt

Check category of both Lookup and Token are adjectives or past participles

Rule: AffectAdjective

(

{AffectLookup.category == adjective, Token.category == VBN}

{AffectLookup.category == adjective, Token.category == JJ}

):tag

-->

:tag.Affect = {kind = :tag.AffectLookup.kind,
category = :tag.AffectLookup.category,
rule = "AffectAdjective"}

copy category and kind values from Lookup to new Affect annotation

Grammar rules: finding triples

- We first create temporary annotations for Person, Organization, Vote, Party, Negatives etc. based on gazetteer lookup, NEs etc.
- We then have a set of rules to combine these into pairs or triples:
 - *<Person, Vote, Party> “Tory Phip admits he voted LibDem”.*
 - *<Party, Affect> “When they get a Tory government they'll be sorry.”*
- We create an annotation “Sentiment” which has the following features:
 - kind = “pro_Labour”, “anti_LibDem”, etc.
 - opinion_holder = “John Smith”, “author” etc.

Identifying the Opinion Holder

- If the opinion holder in the pattern matched is a Person or Organization, we just get the string as the value of `opinion_holder`
- If the opinion holder in the pattern matched is a pronoun, we first find the value of the string of the antecedent and use this as the value of `opinion_holder`
- Currently we only match opinion holders within the same sentence.
- If no explicit opinion holder then we use "author" as the value of `opinion_holder`.
- Later we could look at grabbing the details of the twitterer instead of just using "author".

Grammar rules: finding antecedents

- Find the antecedents of pronouns within a sentence so that we can refer a sentiment back to the original opinion holder or object of the opinion.
- First run the pronominal coreference PR
- Then use a JAPE rule to find pronouns linked to a Person or Organization
- We can identify these because they will have the feature “ENTITY_MENTION_TYPE” (created by the coreferencer)
- The co-referring pronouns all have also an antecedent_offset feature pointing to the proper noun antecedent
- The matching proper noun antecedent is found and its string is added as a feature on the relevant pronoun annotation





Implicit Opinion Holders

- There may not always be an explicit opinion holder
- In many cases, the author of the tweet is the opinion holder
 - *I'm also going to vote Tory. Hello new world.*
 - Here we can co-refer “I” with the person tweeting (using the metadata)
- In other cases, there is no explicit opinion holder:
 - *“Vote for Labour. Harry Potter would.”*
 - However, we can infer by this instruction that the author of the tweet shares this opinion.
- In all these cases, we add the value “author” to the feature “opinion_holder”

Creating the Application

- We only want to process the actual text of the tweet, not all the other information
- To do this, we use a Segment Processing PR to run the sentiment app over just the "text" annotation in Original Markups set.
- So, we need two applications: one containing the Segment Processing PR and one containing the actual sentiment application

Runtime Parameters for the "Segment Processing PR_0001E" Segment Processing PR:

Name	Type	Required	Value
 controller	CorpusController	✓	 twitter app
 inputASName	String		Original markups
 segmentAnnotationType	String	✓	text

Hands-on 1: Analysing political tweets

- Load the ANNIE, Tools and Alignment plugins
- Load the document corpus/politwits-smiley.xml from the hands-on material and add it to a corpus
- Load the application resources/sentiment-all.gapp
- This should also load the sentiment-processing application
- Run sentiment-all.gapp on the corpus and look at the results
- Tip: Make sure you run the right application
- Click on **Sentiment** to see the overall Sentiment of the tweet
 - Sentiment.kind should be **pro-Con**
- Click on **Party** and **Vote** to see intermediate annotations

Hands-on 2: Modifying the application

- Task: modify the application to annotate the following sentence as “anti_Tory”:
 - *They all voted Tory :-（*
- Step 1: add the emoticon as a new entry in the affect gazetteer list `affect_sadness.lst`, with feature “category” and value “smiley”
- Step 2: add a new rule in the `affect.jape` grammar to create an Affect annotation from an AffectLookup with `category=smiley`
- Reinitialise grammar and gazetteer and rerun the application
- **Sentiment.kind** should now be **anti-Con** instead of **pro-Con**



Hands-on 3: Using ANNIC

- Create a new Lucene datastore in GATE
- Use default parameters, except set “AnnotationSets” parameter to exclude “Key”
- Create a new empty corpus, save it to the datastore, then populate it with the politwits-500 corpus
- Run the application on the corpus (this may take a little while)
- Select “Lucene datastore searcher” from the datastore viewer
- Try out some patterns to see what results you get: if you find a pattern that enables you to find an opinion, try implementing it in a JAPE grammar
- If you get stuck, have a look at some examples of patterns on the next slide.....

Hands-on 3: Pattern examples

- Try thinking up your own patterns, but here are some you can try out to get you started (try changing the number of Tokens, or adding negatives):
- `{Lookup.majorType == negation} ({Token})*4 {Lookup.majorType == "vote"}{Lookup.majorType == "party"}`
- `{Token.string == "I"} ({Token})*4 {Lookup.majorType == "vote"}
{Lookup.majorType == "party"}`
- `{Person} ({Token})*4 {Lookup.majorType == "vote"}
{Lookup.majorType == "party"}`
- `{Affect} ({Token})*5 {Lookup.majorType == "candidate"}`
- `{Vote} ({Token})*5 {Lookup.majorType == "candidate"}`

Linguistic information for better analysis

- Linguistic information can give you a lot of clues about meaning
- “Good battery life” seems to indicate a positive feature.
- But conditional sentences can have subtly different meanings:
 - *I'd have bought a Nikon if I'd wanted good battery life*
 - *I'll buy a Nikon if it has good battery life*
 - *I'll buy a Nikon if I want good battery life*
 - *I'd buy a Nikon unless I wanted good battery life*
 - *I'd buy a Nikon even if it doesn't have good battery life.*

Conditional Types

0. If a camera has 20 hours of battery life, you can take many pictures.
 - statement of fact or certainty
1. If someone makes a camera with 20 hours of battery life, I'll buy it
 - potential conditional
 - *long battery life is my top priority*
2. If someone made a camera with 20 hours of battery life, I'd buy it
 - less probably conditional. Indicates preference
 - *as (1), but I think it's unrealistic so I'll settle for something else*
3. If someone had made a camera with 20 hours of battery life, I'd have bought it
 - Impossible past events
 - *as (1), but they don't make one, so I bought something else*

More examples

2. If I wanted a camera with 20 hours of battery life, I would buy a Nikon
 - *battery life is not my priority, so I'll probably buy something else*

3. If I had wanted a camera with 20 hours of battery life, I'd have bought a Nikon
 - *battery life is not my priority and so I bought something other than Nikon*

Linguistic analysis of conditional types

Type 0: If + simple present --> simple present

- *If it has good battery life, you can take lots of pictures*

Type 1: If + simple present --> simple future

- *If it has good battery life, I will buy it*

Type 2: If + past --> would + infinitive

- *If it had good battery life, I would buy it*

Type 3: If + past perfect --> present perfect

- *If it had had good battery life, I would have bought it*

Simple conditional application in GATE

- Gazetteer list gives us words associated with conditionals
 - if, assuming, even if, as long as, on condition that... (positive)
 - unless (negative)
- Verb chunker segments the VPs and also gives
 - the tense of the verb
 - active or passive
 - positive or negative
- Grammar rules combine items from gazetteer with verb information to create rules for sentences

Sample grammar for type 0 conditional

Input: Split VG ConditionalIndicator

Pattern: If + simple present, simple present

Rule: Conditional0

```
(
  {ConditionalIndicator}
  {VG.tense == SimPre}
  {VG.tense == SimPre}
  {Split}
):tag
-->
:tag.Conditional = {type = "0"}
```

{Lookup.majorType == conditional}

Verb phrases with verb in the simple present tense

Don't let the pattern span a sentence boundary

Tag the whole sentence as a conditional of type 0

Why do we do the Lookup in a separate phase?

- Why do we first find the Conditional Lookups and annotate them separately? Why not just use the Lookup annotation within the rule?
- The clue is in the Input headers
- If we use a Lookup annotation within the rule, we need to add “Lookup” to the Input headers
- What effect might this have on the rule?
- Remember that we only want to state explicitly in the rule the things we care about.
- We don't care (at this stage) which nouns occur in the sentence so we want to leave as much as possible unspecified.



Hands-on 3: conditionals

- Remove all loaded applications and documents from GATE
- Load the application resources/conditionals.gapp from the hands-on materials
- Load the text corpus/conditional-sentences.txt, add to a corpus and run the application on it
- Check the results
- Have a look at the grammar conditional-polarity.jape and see if you can work out how the negation part works

Negation: adding the polarity feature

- The sentence is divided into its two verb phrases: firstPol and secondPol
- For each phase, if the value of the neg feature is “yes”, then “neg” is stored as the new value
- If the value of the neg feature is “no”, then “pos” is stored as the new value
- A new feature called “polarity” is added to the final annotation that covers the whole sentence
- The values of the two neg features (one for each VP) are added consecutively as the values of polarity, e.g. “neg” + “pos”

Machine Learning Hands-on Exercise



Machine Learning for Sentiment Analysis

- ML is an effective way to classify opinionated texts
- We want to train a classifier to categorize free text according to the training data.
- Good examples are consumers' reviews of films, products, and suppliers.
- Sites like www.pricegrabber.co.uk show reviews and an overall rating for companies: these make good training and testing data
- We train the ML system on a set of reviews so it can learn good and bad reviews, and then test it on a new set of reviews to see how well it distinguishes between them



Examples of consumer reviews

Merchant Info

Merchant Ratings

Uncategorized Products

Sort Reviews by: Date Rating

[Write a Review »](#)

Date Reviewed: 16/04/08

poet2000

Member Since:

16/04/08

View Member's:

[Reviews](#)

30 days and still waiting

Overall Rating



Date Reviewed: 24/01/07

Dbeach135

Member Since:

24/01/07

View Member's:

[Reviews](#)

Jessops not only failed to complete the next day delivery, the item sent, a digital picture frame did not meet their specification. We ordered it as they claimed on their website that it accepted XD cards. This however was not the case. Jessops felt that they had done nothing wrong although their website was obviously wrong. This incorrect information still is outstanding and they have done nothing to correct their website even though I have notified them of the error.

Overall Rating





Preparing the corpus

- Corpus of 40 documents containing 552 company reviews.
- Each review has a 1- to 5-star rating.
- We pre-processed these in GATE to label each review with a comment annotation with a rating feature
- In ML terms:
 - instance = *comment* annotation
 - class = *rating* feature
 - attributes = NLP features of the underlying text
- We will keep the spans of the comment annotations and use ML to classify them with the *rating* feature



Annotated review

Annotation Sets Annotations List Annotations Stack Co-reference Editor Text

04/01/07

View Member's:
 Reviews Easy to use site. Will visit again. Products delivered very
 speedily. Good value for money. Overall Rating

Key
 comment
 Original markups

Date P
 eo.sm
 Memb
 03/01/

View M
 Review
 with th
 made

Only issue is
 gles. This
 l Rating

C rating 5_Star_Review X
 C X
 ▶ Open Search & Annotate tool



Developing the training application

- We will develop an application that runs a set of NLP components to provide ML instance attributes, and train the classifier
- Take the config file **ml-exercise/paum.xml** from the hands-on materials and copy it to a writable directory on your machine.
- The Batch Learning PR needs to create a **savedFiles** directory and write inside that.
- Load the ANNIE, Tools, and Learning plugins.
- Create a new corpus called “training” and populate it from the directory **ml-exercise/corpora/training** in the hands-on material



Batch learning config (paum.xml)

```
<PARAMETER name="thresholdProbabilityClassification"
            value="0.5" />
```

- This threshold will probably produce a class for each instance
- Classification problems do not use the other threshold probability parameters

```
<multiClassification2Binary method= "one-vs-others" />
```

- this is much faster than one-vs-another

```
<ENGINE nickname="PAUM" implementationName="PAUM"
         options=" -p 50 -n 5 -optB 0.0 " />
```

- Perceptron with uneven margins
- default options

Batch learning config (2)

<INSTANCE-TYPE>comment</INSTANCE-TYPE>

<ATTRIBUTE>

<NAME>Class</NAME>

<SEMTYPE>NOMINAL</SEMTYPE>

<TYPE>comment</TYPE>

<FEATURE>rating</FEATURE>

<POSITION>0</POSITION>

<CLASS/></ATTRIBUTE>

- Takes comment annotations as instances, and classifies them using the rating feature.
- The classes (values of the rating features) form an unordered set (current limitation of the PR).

Batch learning config (3)

```
<NGRAM>  
<NAME>ngram</NAME>  
<NUMBER>1</NUMBER>  
<CONSNUM>1</CONSNUM>  
<CONS-1>  
  <TYPE>Token</TYPE>  
  <FEATURE>root</FEATURE>  
</CONS-1>  
</NGRAM>
```

- Uses unigrams of *Token.root* features inside the comment annotations as the instance attributes (bag of words).
- An additional feature in the hands-on file is commented out for you to experiment with later.



Building the training application (1)

- Create the following PRs with the default init parameters:
 - Document Reset PR
 - Annotation Set Transfer
 - ANNIE English Tokeniser
 - ANNIE Sentence Splitter
 - ANNIE POS Tagger
 - GATE Morphological Analyser
- Create a Batch Learning PR with the configFileURL init parameter set to the new location of your **paum.xml** file
- Create a new Conditional Corpus Pipeline.

Building the application (2)

- We want to copy the comment annotations to the default annotation set to provide the ML instances and classes
- We want to make sure we don't remove the Key annotations
- Add the PRs to the pipeline as follows
 - Document Reset:
 - setsToKeep = "Key"
 - Annotation Set Transfer:
 - annotationTypes = [comment]
 - copyAnnotations = true
 - inputASName = "Key"
 - outputASName, tagASName, textTagName = ""



Building the application (3)

- Add the remaining loaded PRs to the pipeline
 - English tokeniser
 - Sentence splitter
 - POS tagger
 - Morphological analyser
 - Batch Learning:
 - `inputASName, outputASName = ""`
 - `learningMode = TRAINING`
- Run it on the training corpus (this should take less than 1 minute)
- The classifier's model is stored in the **savedFiles** directory beside the **paum.xml** file. The model is stored in text files, but they are not meant to be human-readable.

Applying the training model

- Create a “testing” corpus and populate it from the **corpora/testing** directory.
- To apply the classifier, we need to have comment annotations *without* rating features on the default AS. These will give us the instances to classify. A simple JAPE Transducer can do this.
- Load the grammar resources/grammar/copy_comment_spans.jape_
- Insert the grammar in the pipeline after the AS Transfer PR.
- Set the transducer parameters:
 - inputASName = “Key”
 - outputASName = “”

Applying the training model (2)

- Set the AS Transfer PR's run-mode to “no”
- Set the Batch Learning PR's parameters:
 - `inputASName = ""`
 - `learningMode = APPLICATION`
 - `outputASName = "Output"`
- The classifier will get instances and attributes from the default AS and put instances with classes in the Output AS.
- Run the pipeline on the testing corpus
- Open a few documents and inspect the “comment” annotations:
 - Key set = user ratings
 - default set = instances with no classes
 - Output set = instances with ML classes



Annotation Results

Annotation Sets Annotations List Annotations Stack Co-reference Editor Text

View Member's:
Reviews Where the Hell is it??? It has not arrived yet. will not deal with again Overall Rating

Date Review
tony
Member Since
09/11/05

View Member's:
Reviews God Overall Rating

Date Review
r.a.baxter
Member Since:
29/10/05

View Member's:
Reviews Straight forward experience with no hassle Overall Rating

comment

prob	1.0	X
rating	1_Star_Review	X
		X

Open Search & Annotate tool

- Sentence
- SpaceToken
- Split
- Token
- comment
- Key
 - comment
 - Original markups
 - Output
 - comment

t	Start	End	Id	Features
out	3514	3638	24575	{prob=1.0, rating=5 Star Review}
out	3741	3898	24576	{prob=1.0, rating=5 Star Review}
out	3997	4047	24577	{prob=1.0, rating=5 Star Review}
out	4147	4219	24578	{prob=1.0, rating=1 Star Review}
out	4314	4359	24579	{prob=1.0, rating=5 Star Review}
out	4460	4502	24580	{prob=1.0, rating=5 Star Review}
out	4606	4626	24581	{prob=1.0, rating=5 Star Review}
out	4724	4766	24582	{prob=1.0, rating=5 Star Review}



Evaluation: Precision and Recall

- On the test corpus, click the Corpus Quality Assurance tab.
- Select
 - Annotation Sets A = *Key*, B = *Output*
 - Annotation Types = *comment*
 - Annotation Features = *rating*
 - F-Score = F1.0-score strict
- Click “Compare”
- If every instance has been classified, then the total $P = R = F1$, because every spurious classification is paired with a missing classification
- Use the “Document statistics” sub-pane of Corpus QA to confirm this, and to see how specific documents were annotated



Corpus QA parameters

The screenshot shows a vertical sidebar in the GATE software interface. At the top, there is a toolbar with icons for file operations (G, save, download, undo, refresh). Below the toolbar, the sidebar is divided into several sections:

- Annotation Sets A/Key & [Default set]**
 - Key (A)
 - Original markups
 - Output (B)
 - present in every doc
- Annotation Types**
 - comment
 - present in every sele
- Annotation Features**
 - prob
 - rating
 - present in every sele
- Measures**
 - F-Score
 - Classification
 - F1.0-score strict
 - F1.0-score lenient
 - F1.0-score average
 - F1.0-score strict BDM
- At the bottom, there is a gear icon and the text "Compar".

Results

Corpus statistics		Document statistics					
Annotation	Match	Only A	Only B	Overlap	Rec.B/A	Prec.B/A	F1.0-s.
comment	79	20	20	0	0.80	0.80	0.80
Macro summary					0.80	0.80	0.80
Micro summary	79	20	20	0	0.80	0.80	0.80



Evaluation: Cohen's Kappa

- Click on Measures in the Corpus QA settings and select “Classification”, then Cohen's Kappa, and click “Compare”.
- In addition to the document statistics with summary lines, there is now a “Confusion Matrices” sub-pane.
- The confusion matrix shows how many of each class were classified in that class (or in other classes)
- Cells in these tables can be selected with the mouse (or Ctrl-A to select all) and copied with Ctrl-C, so you can paste them in to a spreadsheet.



Built-in cross-validation

- Cross-validation is a standard way to “stretch” the validity of a manually annotated corpus, because it enables you to test on a larger number of documents
- The 5-fold averaged result is more significant than the result obtained by training on 80% of the same corpus and testing on 20% once.
- In GATE, you can't use the Corpus QA tool on the result, but you can get a detailed statistical report at the end, including P, R, & F1 for each class.

Built-in cross-validation

- The config file includes:
 - `<EVALUATION method="kfold" runs="5" ratio="0.66" />`
 - kfold ignores the ratio setting
 - holdout ignores the runs setting
- The Batch Learning PR will automatically split the corpus into 5 parts, and then
 - train on 1,2,3,4; apply on 5; then
 - train on 1,2,3,5; apply on 4; ...
 - train on 2,3,4,5; apply on 1;
 - and average the results.

Built-in cross-validation

- The application can be modified slightly to perform cross-validation according to the config file
- Switch the AS Transfer PR back on (so it copies the comment annotation to the default AS)
- Switch off the JAPE transducer
- Set the Batch Learning PR parameters:
 - `inputAS, outputAS = ""`
 - `learningMode = EVALUATION`
- Create a new corpus "all" and populate it from the **corpora/all** directory (all the documents from training and testing.)
- Run the pipeline on the new corpus. This will take a few minutes.
- Click on the Messages pane to view the results

Results

*** Averaged results for each label over 5 runs as:

Results of single label:

0 LabelName=1_Star_Review, number of instances=27

(correct, partialCorrect, spurious, missing)=(1.6, 0.0, 0.6, 5.2); (precision, recall, F1)=(0.4666667, 0.25333333, 0.3277778); Lenient: (0.4666667, 0.25333333, 0.3277778)

1 LabelName=2_Star_Review, number of instances=6

(correct, partialCorrect, spurious, missing)=(0.0, 0.0, 0.0, 1.6); (precision, recall, F1)=(0.0, 0.0, 0.0); Lenient: (0.0, 0.0, 0.0)

2 LabelName=3_Star_Review, number of instances=17

(correct, partialCorrect, spurious, missing)=(0.8, 0.0, 0.0, 3.6); (precision, recall, F1)=(0.4, 0.24000001, 0.26666665); Lenient: (0.4, 0.24000001, 0.26666665)

3 LabelName=4_Star_Review, number of instances=92

(correct, partialCorrect, spurious, missing)=(1.8, 0.0, 3.4, 21.2); (precision, recall, F1)=(0.44666666, 0.0984191, 0.1436075); Lenient: (0.44666666, 0.0984191, 0.1436075)

4 LabelName=5_Star_Review, number of instances=298

(correct, partialCorrect, spurious, missing)=(72.8, 0.0, 29.4, 1.8); (precision, recall, F1)=(0.71384054, 0.9741704, 0.819043); Lenient: (0.71384054, 0.9741704, 0.819043)

Overall results as:

(correct, partialCorrect, spurious, missing)=(77.0, 0.0, 33.4, 33.4); (precision, recall, F1)=(0.6971326, 0.6971326, 0.6971326); Lenient: (0.6971326, 0.6971326, 0.6971326)

This learning session finished!

Looking into the future

- Typically, opinion mining looks at social media content to analyse people's explicit opinions about a product or service
- This backwards-looking approach often aims primarily at dealing with problems, e.g. unflattering comments
- A forwards-looking approach aims at looking ahead to understanding potential new needs from consumers
- This is not just about looking at specific comments, e.g. “the product would be better if it had longer battery life”, but also about detecting non-specific sentiment
- This is achieved by understanding people's needs and interests in a more general way, e.g. drawing conclusions from their opinions about other products, services and interests.

The problem of sparse data

- One of the difficulties of drawing conclusions from traditional opinion mining techniques is the sparse data issue
- Opinions tend to be based on a very specific product or service, e.g. a particular model of camera, but don't necessarily hold for every model of that brand of camera, or for every product sold by the company
- One solution is figuring out which statements can be generalised to other models/products and which are specific
- Another solution is to leverage sentiment analysis from more generic expressions of motivation, behaviour, emotions and so on, e.g. what type of person buys what kind of camera?

Predicting future behaviour

- Social media provides a wealth of information about a user's behaviour and interests, from the explicit “John's interests are tennis, swimming and classical music”, the implicit “people who like skydiving tend to be big risk-takers” and the associative “people who buy Nike products also tend to buy Apple products”
- While information about individuals isn't useful on its own, finding defined clusters of interests and opinions is
- For example, if many people talk on social media sites about fears in airline security, life insurance companies might consider opportunities to sell a new service
- This kind of predictive analysis is all about understanding your potential audience at a much deeper level - this can lead to improved advertising techniques such as personalised ads to different groups



Summary

- Introduced the concept of Opinion Mining and Sentiment Analysis
- Simple examples of rule-based and ML methods for creating OM applications
- Examples of how deeper linguistic information can be useful
- Practice with complex applications
- Looking ahead to the future

Suggestions for
further ML experiments...



Suggestions...

- The config file URL is an init parameter, but the contents can be re-loaded, so you can
 - use any text editor on the config file, save the changes, and
 - re-initialize the Batch Learning PR to re-load the file with changes.



Suggestions...

- Try n-grams where $n > 1$
 - Change `<NUMBER>` in the config
 - Usually this is slower, but sometimes it improves quality
- combining features
 - change `<CONSUM>` in the config to 2 and uncomment the *Token.orth* element
 - this concatenates the features



Suggestions...

- Adjust the `thresholdProbabilityClassification`
 - Increasing it may increase precision and decrease recall, and may prevent the classifier from assigning a class to every instance.
 - Decreasing it may increase recall and decrease precision.
 - This is the “pickiness” control of the classifier.



Suggestions...

- Try using other features
 - *Token.string*, *Token.category*, or combinations of these with *Token.root* and *Token.orth*
- You could even include other ANNIE PRs in the pipeline and use Lookup or other annotation types.
 - You need to run the same annotation-creating PRs for training and application.
 - If the config file specifies an annotation that is missing in an instance, the ML PR will throw an exception.