In [1]:
```
import gzip
import math
import matplotlib.pyplot as plt
import numpy
import random
import sklearn
import string
from collections import defaultdict
from gensim.models import Word2Vec
from nltk.stem.porter import *
from sklearn import linear_model
from sklearn.manifold import TSNE
```

Data is available at http://cseweb.ucsd.edu/~jmcauley/pml/data/. Download and save to your own directory

In [2]:
```
dataDir = "/home/jmcauley/pml_data/"
```

Bag-of-words models

In [3]:
```
def parseData(fname):
    for l in open(fname):
        yield eval(l)
```

In [4]:
```
data = list(parseData(dataDir + "beer_50000.json"))[:5000]
```

How many unique words are there?

In [5]:
```
wordCount = defaultdict(int)
for d in data:
    for w in d['review/text'].split():
        wordCount[w] += 1

len(wordCount)
```

Out[5]:
```
36225
```

Ignore capitalization and remove punctuation

In [6]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    for w in r.split():
        wordCount[w] += 1

len(wordCount)
```

Out[6]:
```
19426
```

With stemming

In [7]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
stemmer = PorterStemmer()
for d in data:
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  for w in r.split():
    w = stemmer.stem(w)
    wordCount[w] += 1

len(wordCount)
```

Out[7]:
```
14847
```

Just build our feature vector by taking the most popular words (lowercase, punctuation removed, but no stemming)

In [8]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data:
  r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
  for w in r.split():
    wordCount[w] += 1

counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()
```

In [9]:
```
words = [x[1] for x in counts[:1000]]
```

Sentiment analysis

In [10]:
```
wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
```

In [11]:
```
def feature(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation
    for w in r.split():
        if w in words:
            feat[wordId[w]] += 1
    feat.append(1) # offset
    return feat
```

Extract bag-of-word features. For a bigger dataset, replace this with a sparse matrix to save memory (see examples in Chapter 6)

In [12]:
```
X = [feature(d) for d in data]
y = [d['review/overall'] for d in data]
```

In [13]:
```
# Regularized regression
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

N-grams

Simple example...

In [14]:
```
sentence = "Dark red color, light beige foam"
unigrams = sentence.split()
bigrams = list(zip(unigrams[:-1], unigrams[1:]))
trigrams = list(zip(unigrams[:-2], unigrams[1:-1], unigrams[2:]))
```

In [15]:
```
trigrams
```

Out[15]:
```
[('Dark', 'red', 'color,'),
 ('red', 'color,', 'light'),
 ('color,', 'light', 'beige'),
 ('light', 'beige', 'foam')]
```

Extract n-grams up to length 5 (same dataset as example above)

In [16]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in data:
    r = ''.join([c for c in d['review/text'].lower() if not c in punctuation])
    ws = r.split()
    ws2 = [' '.join(x) for x in list(zip(ws[:-1],ws[1:]))]
    ws3 = [' '.join(x) for x in list(zip(ws[:-2],ws[1:-1],ws[2:]))]
    ws4 = [' '.join(x) for x in list(zip(ws[:-3],ws[1:-2],ws[2:-1],ws[3:]))]
    ws5 = [' '.join(x) for x in list(zip(ws[:-4],ws[1:-3],ws[2:-2],ws[3:-1],ws[4
    for w in ws + ws2 + ws3 + ws4 + ws5:
        wordCount[w] += 1

counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()
```

In [17]:
```
words = [x[1] for x in counts[:1000]]
```

A few of our 1000 most popular n-grams. Note the combination of n-grams of different lengths

In [18]:
```
words[200:210]
```

Out[18]:
```
['pint',
 'hoppy',
 'stout',
 'though',
 'lots',
 'and the',
 'malty',
 'mouthfeel is',
 'even',
 'quickly']
```

In [19]:

```
wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
```

In [20]:

```
def feature(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review/text'].lower() if not c in punctuation
    ws = r.split()
    ws2 = [' '.join(x) for x in list(zip(ws[:-1],ws[1:]))]
    ws3 = [' '.join(x) for x in list(zip(ws[:-2],ws[1:-1],ws[2:]))]
    ws4 = [' '.join(x) for x in list(zip(ws[:-3],ws[1:-2],ws[2:-1],ws[3:]))]
    ws5 = [' '.join(x) for x in list(zip(ws[:-4],ws[1:-3],ws[2:-2],ws[3:-1],ws[4
    for w in ws + ws2 + ws3 + ws4 + ws5:
        if w in words:
            feat[wordId[w]] += 1
    feat.append(1) #offset
    return feat
```

Same as the model in the previous example above, except using n-grams rather than just unigrams

In [21]:

```
X = [feature(d) for d in data]
y = [d['review/overall'] for d in data]
```

In [22]:

```
clf = linear_model.Ridge(1.0, fit_intercept=False) # MSE + 1.0 l2
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

In [23]:

```
wordSort = list(zip(theta[:-1], words))
wordSort.sort()
```

Some of the most negative and positive n-grams...

In [24]:

```
wordSort[:20]
```

Out[24]:

```
[(-0.39143632655636557, 'a lot of'),
 (-0.28212293516954884, 'a bit more'),
 (-0.2661141170411719, 'water'),
 (-0.2530229314150494, 'the background'),
 (-0.24105277280700568, 'corn'),
 (-0.21580888155062322, 'as it warms'),
 (-0.20470888977263463, 'hint of'),
 (-0.2032403642895844, 'yellow'),
 (-0.19776086681438282, 'little more'),
 (-0.19515009289478227, 'watery'),
 (-0.1833549486926261, 'straw'),
 (-0.1784682882644964, 'a hint'),
 (-0.1749628781772528, 'around the'),
 (-0.17387445409086819, 'kind'),
 (-0.17381459136980834, 'down the'),
 (-0.1723151420771276, 'lot'),
 (-0.16887649424485385, 'weak'),
 (-0.16647751584896267, 'bad'),
 (-0.16513650187976975, 'old'),
 (-0.1581538428300185, 'up to')]
```

In [25]:

```
wordSort[-20:]
```

Out[25]:

```
[(0.1596617274119243, 'wonderful'),
 (0.16316670449503792, 'off white head'),
 (0.17379190414356988, 'a little more'),
 (0.17560247184995886, 'favorite'),
 (0.17674628573616408, 'not too'),
 (0.17724972085645002, 'i really'),
 (0.17919486420127723, 'easy to'),
 (0.18200914314264174, 'i am'),
 (0.18394566780123306, 'background'),
 (0.18752400440800757, 'touch of'),
 (0.19751810949893422, 'the best'),
 (0.19933817230459436, 'hint'),
 (0.21249123771083625, 'this one is'),
 (0.2141045147587393, 'a hint of'),
 (0.22043777880133708, 'not bad'),
 (0.2376891974695213, 'of these'),
 (0.23871898220027807, 'a bad'),
 (0.24874277211071016, 'lot of'),
 (0.2748653479324936, 'a lot'),
 (0.3121741564883301, 'bit more')]
```

TF-IDF

In [26]:

```
dataset = []
```

Small set of Goodreads fantasy reviews

In [27]:

```
z = gzip.open(dataDir + "goodreads_reviews_fantasy_paranormal.json.gz")
for l in z.readlines():
    dataset.append(eval(l))
    if len(dataset) == 50000:
        break
```

For example...

In [28]:

```
dataset[0]
```

Out[28]:

```
{'book_id': '18245960',
 'date_added': 'Sun Jul 30 07:44:10 -0700 2017',
 'date_updated': 'Wed Aug 30 00:00:26 -0700 2017',
 'n_comments': 1,
 'n_votes': 28,
 'rating': 5,
 'read_at': 'Sat Aug 26 12:05:52 -0700 2017',
 'review_id': 'dfdbb7b0eb5a7e4c26d59a937e2e5feb',
 'review_text': 'This is a special book. It started slow for about the first thi
rd, then in the middle third it started to get interesting, then the last third
blew my mind. This is what I love about good science fiction - it pushes your th
inking about where things can go. \n It is a 2015 Hugo winner, and translated fr
om its original Chinese, which made it interesting in just a different way from
most things I\'ve read. For instance the intermixing of Chinese revolutionary hi
story - how they kept accusing people of being "reactionaries", etc. \n It is a
book about science, and aliens. The science described in the book is impressive
- its a book grounded in physics and pretty accurate as far as I could tell. Tho
ugh when it got to folding protons into 8 dimensions I think he was just making
stuff up - interesting to think about though. \n But what would happen if our SE
TI stations received a message - if we found someone was out there - and the per
son monitoring and answering the signal on our side was disillusioned? That part
of the book was a bit dark - I would like to think human reaction to discovering
alien civilization that is hostile would be more like Enders Game where we would
band together. \n I did like how the book unveiled the Trisolaran culture throug
h the game. It was a smart way to build empathy with them and also understand wh
at they\'ve gone through across so many centuries. And who know a 3 body problem
was an unsolvable math problem? But I still don\'t get who made the game - maybe
that will come in the next book. \n I loved this quote: \n "In the long history
of scientific progress, how many protons have been smashed apart in accelerators
by physicists? How many neutrons and electrons? Probably no fewer than a hundred
million. Every collision was probably the end of the civilizations and intellige
nces in a microcosmos. In fact, even in nature, the destruction of universes mus
t be happening at every second--for example, through the decay of neutrons. Als
o, a high-energy cosmic ray entering the atmosphere may destroy thousands of suc
h miniature universes...."',
 'started_at': 'Tue Aug 15 13:23:18 -0700 2017',
 'user_id': '8842281e1d1347389f2ab93d60773d4d'}
```

Similar process to extract bag-of-words representations as in previous examples

In [29]:

```python
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in dataset:
    r = ''.join([c for c in d['review_text'].lower() if not c in punctuation])
    for w in r.split():
        wordCount[w] += 1
```

In [30]:

```python
counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()
```

In [31]:

```python
words = [x[1] for x in counts[:1000]]
```

Document frequency (df)

In [32]:

```python
df = defaultdict(int)
for d in dataset:
    r = ''.join([c for c in d['review_text'].lower() if not c in punctuation])
    for w in set(r.split()):
        df[w] += 1
```

Term frequency (tf)

Here we extract frequencies for terms in a single specific review

In [33]:
```
rev = dataset[9] # Query review
rev
```

Out[33]:
```
{'book_id': '76620',
 'date_added': 'Tue Sep 02 17:03:40 -0700 2008',
 'date_updated': 'Wed Dec 14 12:30:43 -0800 2016',
 'n_comments': 2,
 'n_votes': 30,
 'rating': 5,
 'read_at': 'Tue May 05 00:00:00 -0700 2015',
 'review_id': '9206654986a733bd753753aa0c882813',
 'review_text': "I read this after hearing from a few people that it was among t
heir all-time favorites. I was almost put off when I saw it was a story about ra
bbits, originally written as a tale by a father to his children - but I'm glad I
wasn't. \n I found the folk tales about El-ahrairah to be very impressive. The a
uthor clearly had a vivid imagination to create so much of the rabbits culture a
nd history. But I think this book was worth reading as it's really a story about
survival, leadership, and human nature. \n Oh and Fiver rocks. And BigWig is the
man.",
 'started_at': 'Sun Apr 19 00:00:00 -0700 2015',
 'user_id': '8842281e1d1347389f2ab93d60773d4d'}
```

In [34]:
```
tf = defaultdict(int)
r = ''.join([c for c in rev['review_text'].lower() if not c in punctuation])
for w in r.split():
    # Note = rather than +=, different versions of tf could be used instead
    tf[w] = 1

tfidf = dict(zip(words,[tf[w] * math.log2(len(dataset) / df[w]) for w in words]))
tfidfQuery = [tf[w] * math.log2(len(dataset) / df[w]) for w in words]
```

Find the highest tf-idf words in our example review

In [35]:
```
maxTf = [(tf[w],w) for w in words]
maxTf.sort(reverse=True)
maxTfIdf = [(tfidf[w],w) for w in words]
maxTfIdf.sort(reverse=True)
```

In [36]:
```
maxTfIdf[:10]
```

Out[36]:
```
[(6.475214154215886, 'nature'),
 (6.1604918290613755, 'tales'),
 (5.77675046027207, 'children'),
 (5.7117950184313, 'saw'),
 (5.316168825598678, 'history'),
 (5.286585714095165, 'father'),
 (4.98210658996402, 'worth'),
 (4.910068021149542, 'glad'),
 (4.881720019873613, 'human'),
 (4.731589561708275, 'tale')]
```

Cosine similarity

In [37]:

```
def Cosine(x1,x2):
    numer = 0
    norm1 = 0
    norm2 = 0
    for a1,a2 in zip(x1,x2):
        numer += a1*a2
        norm1 += a1**2
        norm2 += a2**2
    if norm1*norm2:
        return numer / math.sqrt(norm1*norm2)
    return 0
```

Find the other reviews in the corpus with the highest cosine similarity between tf-idf vectors

In [38]:

```
similarities = []
for rev2 in dataset:
    tf = defaultdict(int)
    r = ''.join([c for c in rev2['review_text'].lower() if not c in punctuation])
    for w in r.split():
        # Note = rather than +=
        tf[w] = 1
    tfidf2 = [tf[w] * math.log2(len(dataset) / df[w]) for w in words]
    similarities.append((Cosine(tfidfQuery, tfidf2), rev2['review_text']))
```

In [39]:

```
similarities.sort(reverse=True)
```

In [40]:

```
similarities[:10]
```

Out[40]:

```
[(1.0,
  "I read this after hearing from a few people that it was among their all-time
favorites. I was almost put off when I saw it was a story about rabbits, origina
lly written as a tale by a father to his children - but I'm glad I wasn't. \n I
found the folk tales about El-ahrairah to be very impressive. The author clearly
had a vivid imagination to create so much of the rabbits culture and history. Bu
t I think this book was worth reading as it's really a story about survival, lea
dership, and human nature. \n Oh and Fiver rocks. And BigWig is the man."),
 (0.33623828530476574,
  'Need a review \n For a full explanation of the provenance and history of Arab
ian Nights, as well as an alternative translation, see \n https://www.theatlanti
c.com/entertain... \n For children, see \n Aladdin and Other Tales from the Arab
ian Nights, Dawood, 1960 \n https://www.goodreads.com/review/show...'),
 (0.2944751254505903,
  'Awesome! Love this author and his story world. Found quite a few errors/typos
in the text, but the story is worth reading.'),
 (0.2944520286327877,
  "At first I was iffy about this book but it intrigued by the viewpoint it was
written from then before I knew it is was sucked in and couldn't put it down!! I
almost cried when I saw there was 3 more books."),
 (0.2902791374813246,
  "I was delighted by this book ...the only fault is that it was too short ! Wha
t a fantastic idea; a refuge for the children who have had adventures & now cann
ot fit back into the identity assigned to them. How many of us are not comfortab
le in the families we were born to? I loved the way the different doorways were
sorted; one would think that adventures shared would be a bonding moment. Rivalr
ies will be ever present ; guess that is human nature. I don't want to describe
too much & ruin the magic; this author is the heir apparent to Neil Gaiman. Im s
o glad she has more books to discover !"),
 (0.27867963900816956,
  "Loved this tale and look forward to reading much more of Gem Sivad's tale
s."),
 (0.2755108699963019,
  "I'm really liking these books. It was good to learn more about the vampire lo
re in these ones and I'm glad that Jane found some companionship in this book, b
ut I wasn't liking the kidnapped children storyline."),
 (0.2685687782658902,
  'An amazing story. The first taste of the Tolkien universe, this children tale
s has it all and is superbly written. A must absolutely read.'),
 (0.260579392737578,
  "I'm enjoying reading this series. I wish I would have read this before I saw
the movie...but oh well."),
 (0.2602744050799704,
  "See full review for Enchanted at: https://toomanybooksnotenoughshelves.... \n
Sunday Woodcutter is the seventh daughter of a seventh daughter. Because she is
such a thing, she has magical properties. Her gift bestowed upon her by her Fair
y Godmother Joy is that whatever she writes comes true. It has had harsh implica
tions in the past, so Sunday just writes about her family history. She had six o
lder sisters and three older brothers. Jack Junior died tragically thanks in lar
ge part to the royal family of Arilland. Monday was married off to a prince and
they never see her anymore. Tuesday died. Wednesday is a little bit strange. Thu
rsday married a pirate king and sends them gifts whenever she can. Friday has a
pure heart and a talent with needle and thread and uses it to make clothes for t
he poor children. Saturday is tomboyish and spends her days with her father and
brother in the wood with her axe. Sunday is very lonely, until she meets an ench
anted frog who she fasts befriends. \n But the irony - of course - is that the f
rog (Grumble) is actually the crown prince Rumbold. Thanks to one of her Sunda
y's kisses, he becomes his human self without any remembrance of his year prior
to becoming a frog. All he knows is he is desperately in love with Sunday and ne
eds her to be his at any cost. So, like in most fairy tales, a few balls are hel
d with all the maidens in the kingdom invited. But maybe they aren't meant to li
ve happily ever after... \n As I've said in my previous posts, I really like fai
ry-tale retellings and this book seemed to be right up my alley. But...it wasn't
to be.")]
```

word2vec (gensim)

A few utility data structures (used later)

In [41]:
```python
beerStyles = {} # Style of each item
categories = set() # Set of item categories
reviewsPerUser = defaultdict(list)
beerIdToName = {} # Map an ID to the name of the product
```

In [42]:
```python
z = open(dataDir + "beer_50000.json")

reviews = []
reviewDicts = []

for l in z:
    d = eval(l)
    reviews.append(d['review/text'])
    beerStyles[d['beer/beerId']] = d['beer/style']
    categories.add(d['beer/style'])
    beerIdToName[d['beer/beerId']] = d['beer/name']
    reviewsPerUser[d['user/profileName']].append((d['review/timeUnix'], d['beer/
    reviewDicts.append(d)
    if len(reviews) == 50000:
        break
```

Tokenize the reviews, so that each review becomes a list of words

In [43]:
```
reviewTokens = []
```

In [44]:
```
punctuation = set(string.punctuation)
for r in reviews:
    r = ''.join([c for c in r.lower() if not c in punctuation])
    tokens = []
    for w in r.split():
        tokens.append(w)
    reviewTokens.append(tokens)
```

Example of a tokenized review

In [45]:
```
reviewTokens[0]
```

Out[45]:
```
['a',
 'lot',
 'of',
 'foam',
 'but',
 'a',
 'lot',
 'in',
 'the',
 'smell',
 'some',
 'banana',
 'and',
 'then',
 'lactic',
 'and',
 'tart',
 'not',
 'a',
 'good',
 'start',
 'quite',
 'dark',
 'orange',
 'in',
 'color',
 'with',
 'a',
 'lively',
 'carbonation',
 'now',
 'visible',
 'under',
 'the',
 'foam',
 'again',
 'tending',
 'to',
 'lactic',
 'sourness',
 'same',
 'for',
 'the',
 'taste',
 'with',
 'some',
 'yeast',
 'and',
 'banana']
```

Fit the word2vec model

In [46]:
```
model = Word2Vec(reviewTokens,
                 min_count=5, # Words/items with fewer instances are discarded
                 size=10, # Model dimensionality
                 window=3, # Window size
                 sg=1) # Skip-gram model
```

Extract word representation for a particular word

In [47]:
```
model.wv['yeast']
```

Out[47]:
```
array([ 0.2489907 , -0.08647301, -0.6366861 ,  0.45150977, -0.922045  ,
       -0.3455544 , -0.7728421 ,  0.8667233 ,  0.01321169, -1.0097654 ],
      dtype=float32)
```

Find similar words to a given query

In [48]:
```
model.wv.similar_by_word("grassy")
```

Out[48]:
```
[('piny', 0.990902841091156),
 ('citrus', 0.9905515909194946),
 ('floral', 0.989111602306366),
 ('citrussy', 0.9866609573364258),
 ('citric', 0.9859722256660461),
 ('flowery', 0.9844235181808472),
 ('pine', 0.9834136962890625),
 ('piney', 0.9828003644943237),
 ('spicy', 0.9817010760307312),
 ('herbal', 0.9794316291809082)]
```

item2vec

Almost the same as word2vec, but "documents" are made up of sequences of item IDs rather than words

In [49]:
```
reviewLists = []
for u in reviewsPerUser:
    rl = list(reviewsPerUser[u])
    rl.sort()
    reviewLists.append([x[1] for x in rl])
```

In [50]:

```
model10 = Word2Vec(reviewLists,
                   min_count=5, # Words/items with fewer instances are discarded
                   size=10, # Model dimensionality
                   window=3, # Window size
                   sg=1) # Skip-gram model
```

In [51]:

```
beerIdToName['7360']
```

Out[51]:

```
'Crystal Bitter Ale'
```

In [52]:

```
model10.wv.similar_by_word('7360')
```

Out[52]:

```
[('55890', 0.9990268349647522),
 ('10788', 0.9987408518791199),
 ('32374', 0.9984040260314941),
 ('4370', 0.9983484745025635),
 ('18898', 0.998065710067749),
 ('48047', 0.9975799918174744),
 ('39606', 0.9974957704544067),
 ('10480', 0.9967398643493652),
 ('36376', 0.9967268109321594),
 ('13009', 0.996677041053772)]
```

In [53]:

```
for b in model10.wv.similar_by_word('7360'):
    print(beerIdToName[b[0]])
```

```
Summerbrau Lager
Pilot Rock Porter
Puget Sound Vanilla Porter
Pinnacle Peak Pale Ale
Ferdinand Svetly Lezak
Matso's Monsoonal Blonde
Leviathan Imperial Stout
Ter Dolen Tripel
La DÃ©livrance
Pale Ale
```

t-SNE embedding

Visualize the embeddings from the model above using t-SNE

In [54]:

```
X = []
beers = []
for b in beerIdToName:
    try:
        X.append(list(model10.wv[b]))
        beers.append(b)
    except Exception as e:
        pass
```

Fit a model with just two components for the sake of visualization

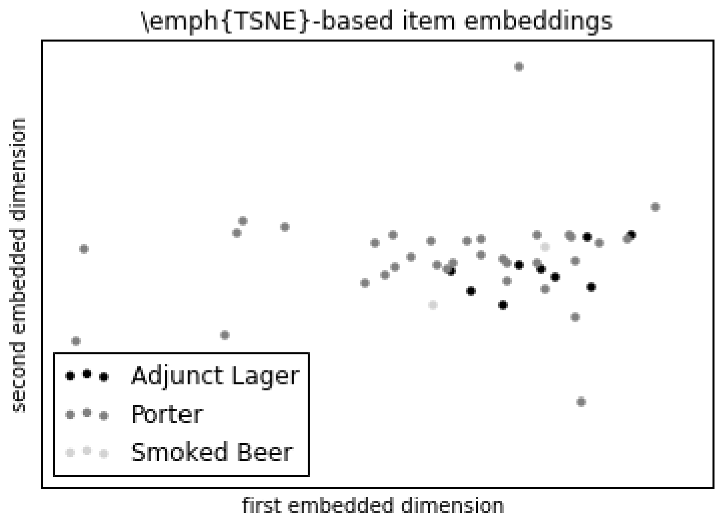
In [55]:
```
X_embedded = TSNE(n_components=2).fit_transform(X)
```

Generate scatterplots using the embedded points (one scatter plot per category)

In [56]:
```
scatterPlotsX = defaultdict(list)
scatterPlotsY = defaultdict(list)
for xy, b in zip(X, beers):
    cat = beerStyles[b]
    try:
        scatterPlotsX[cat].append(xy[0])
        scatterPlotsY[cat].append(xy[1])
    except Exception as e:
        pass
```

Plot data from a few categories (more interesting with a larger dataset)

In [57]:
```
plt.scatter(scatterPlotsX['American Adjunct Lager'],
            scatterPlotsY['American Adjunct Lager'], color='k', lw=0, label="Adj
plt.scatter(scatterPlotsX['American Porter'],
            scatterPlotsY['American Porter'], color='grey', lw=0, label = "Porte
plt.scatter(scatterPlotsX['Smoked Beer'],
            scatterPlotsY['Smoked Beer'], color='lightgrey', lw = 0, label = "Sm
plt.legend(loc='lower left')
plt.xticks([])
plt.yticks([])
plt.xlabel("first embedded dimension ")
plt.ylabel("second embedded dimension")
plt.title("\emph{TSNE}-based item embeddings")
plt.show()
```



In [ ]:

Exercises

8.1 / 8.2

Simple sentiment analysis pipeline

In [58]:
```
def parseData(fname):
    for l in open(fname):
        yield eval(l)
```

In [59]:
```
data = list(parseData(dataDir + "beer_50000.json"))[:5000]
```

In [60]:
```
random.shuffle(data)
```

Add a couple of "options" for the representation (in this case whether we should convert to lower case, remove puncuation). More could be added.

In [61]:
```
def feature(datum, words, wordId, tolower=True, removePunct=True):
    feat = [0]*len(words)
    r = datum['review/text']
    if tolower:
        r = r.lower()
    if removePunct:
        r = ''.join([c for c in r if not c in punctuation])
    for w in r.split():
        if w in words:
            feat[wordId[w]] += 1
    feat.append(1) # offset
    return feat
```

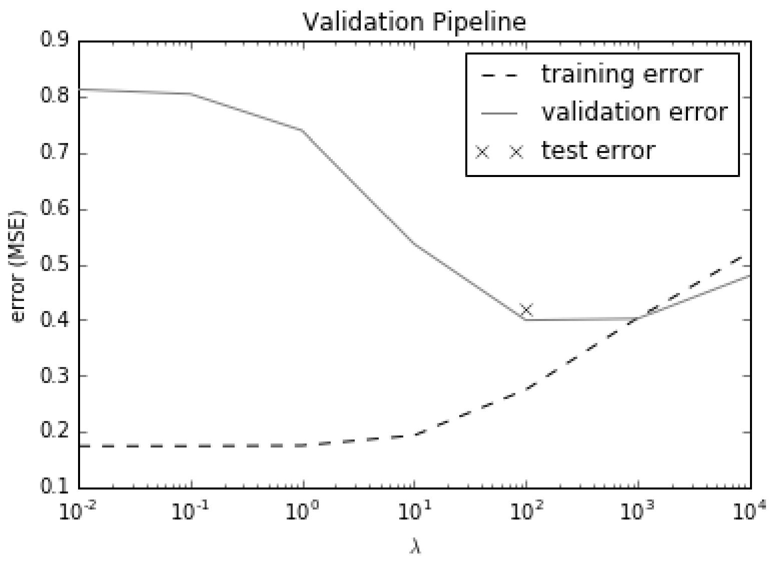
Condense the pipeline code (see Chapter 3) into a single function

In [62]:
```
def pipeline(dSize = 1000, tolower=True, removePunct=True):
    wordCount = defaultdict(int)
    punctuation = set(string.punctuation)
    for d in data: # Strictly, should just use the *training* data to extract wo
        r = d['review/text']
        if tolower:
            r = r.lower()
        if removePunct:
            r = ''.join([c for c in r if not c in punctuation])
        for w in r.split():
            wordCount[w] += 1

    counts = [(wordCount[w], w) for w in wordCount]
    counts.sort()
    counts.reverse()

    words = [x[1] for x in counts[:dSize]]

    wordId = dict(zip(words, range(len(words))))
    wordSet = set(words)

    X = [feature(d, words, wordId, tolower, removePunct) for d in data]
    y = [d['review/overall'] for d in data]
    Ntrain,Nvalid,Ntest = 4000,500,500
    Xtrain,Xvalid,Xtest = X[:Ntrain],X[Ntrain:Ntrain+Nvalid],X[Ntrain+Nvalid:]
    ytrain,yvalid,ytest = y[:Ntrain],y[Ntrain:Ntrain+Nvalid],y[Ntrain+Nvalid:]

    bestModel = None
    bestVal = None
    bestLamb = None

    ls = [0.01, 0.1, 1, 10, 100, 1000, 10000]
    errorTrain = []
    errorValid = []

    for l in ls:
        model = sklearn.linear_model.Ridge(l)
        model.fit(Xtrain, ytrain)
        predictTrain = model.predict(Xtrain)
        MSEtrain = sum((ytrain - predictTrain)**2)/len(ytrain)
        errorTrain.append(MSEtrain)
        predictValid = model.predict(Xvalid)
        MSEvalid = sum((yvalid - predictValid)**2)/len(yvalid)
        errorValid.append(MSEvalid)
        print("l = " + str(l) + ", validation MSE = " + str(MSEvalid))
        if bestVal == None or MSEvalid < bestVal:
            bestVal = MSEvalid
            bestModel = model
            bestLamb = l

    predictTest = bestModel.predict(Xtest)
    MSEtest = sum((ytest - predictTest)**2)/len(ytest)
    MSEtest

    plt.xticks([])
    plt.xlabel(r"$\lambda$")
    plt.ylabel(r"error (MSE)")
    plt.title(r"Validation Pipeline")
    plt.xscale('log')
    plt.plot(ls, errorTrain, color='k', linestyle='--', label='training error')
    plt.plot(ls, errorValid, color='grey',zorder=4,label="validation error")
    plt.plot([bestLamb], [MSEtest], linestyle='', marker='x', color='k', label="
    plt.legend(loc='best')
    plt.show()
```

Run models with different feature representation options and dictionary sizes

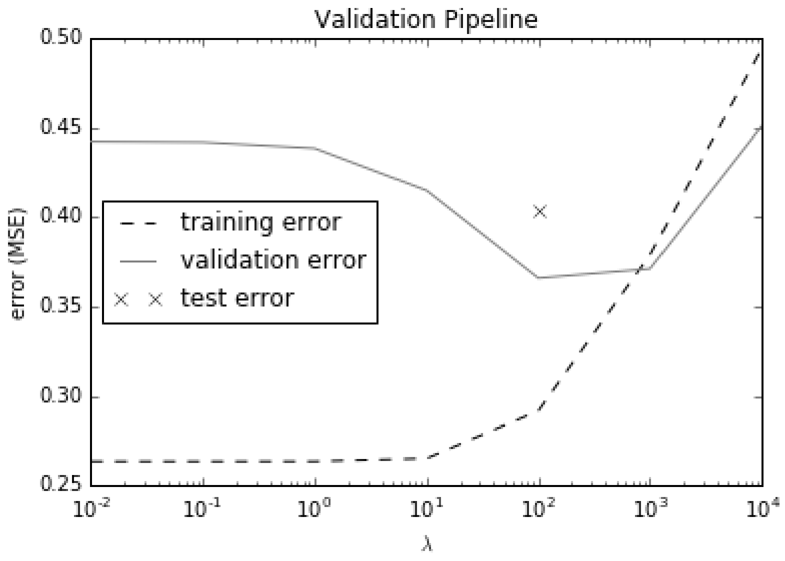
In [63]:
```
pipeline(2000, False, False)
```

```
l = 0.01, validation MSE = 0.8124083444098931
l = 0.1, validation MSE = 0.8041193851382906
l = 1, validation MSE = 0.7387154350328736
l = 10, validation MSE = 0.5361335778700643
l = 100, validation MSE = 0.3996322733152027
l = 1000, validation MSE = 0.40242091430095334
l = 10000, validation MSE = 0.4792032394453763
```



In [64]:
```
pipeline(1000, True, True)
```

```
l = 0.01, validation MSE = 0.4421864459799068
l = 0.1, validation MSE = 0.441830005249123
l = 1, validation MSE = 0.4384260891567443
l = 10, validation MSE = 0.41488435972033527
l = 100, validation MSE = 0.36584997844103806
l = 1000, validation MSE = 0.3710655728890064
l = 10000, validation MSE = 0.4509858498087078
```



8.3

Use item2vec to make recommendations (following code from exercises in Chapter 4)

In [65]:
```
def recScore(i, userHistory):
    historyInVocab = [w for w in userHistory if w in model10.wv]
    if len(historyInVocab) == 0:
        return 0
    sc = model10.wv.distance(str(i), historyInVocab[-1])
    return sc

def rec(userHistory):
    historyInVocab = [w for w in userHistory if w in model10.wv]
    if len(historyInVocab) == 0:
        return 0
    return model10.wv.most_similar(positive = historyInVocab, topn=10)
```

In [66]:

```
recScore(20539, reviewLists[0])
```

Out[66]:

```
0.20018476247787476
```

In [67]:

```
rec(reviewLists[0])
```

Out[67]:

```
[('41688', 0.9940300583839417),
 ('44596', 0.9929537773132324),
 ('24082', 0.9924867749214172),
 ('26393', 0.9917975068092346),
 ('6824', 0.9912790060043335),
 ('8950', 0.9910286068916321),
 ('24473', 0.9906283020973206),
 ('8679', 0.9903963208198547),
 ('5972', 0.990136981010437),
 ('25918', 0.989347517490387)]
```

8.4

(see tf-idf retrieval examples above)

8.5

Predict the rating using item2vec item similarity scores. Adapts models from Chapter 4.

In [68]:

```
ratingMean = sum([d['review/overall'] for d in reviewDicts]) / len(reviewDicts)
```

In [69]:

```
itemAverages = defaultdict(list)
reviewsPerUser = defaultdict(list)

for d in reviewDicts:
    i = d['beer/beerId']
    u = d['user/profileName']
    itemAverages[i].append(d['review/overall'])
    reviewsPerUser[u].append(d)

for i in itemAverages:
    itemAverages[i] = sum(itemAverages[i]) / len(itemAverages[i])
```

In [70]:

```
def predictRating(user,item):
    ratings = []
    similarities = []
    if not str(item) in model10.wv:
        return ratingMean
    for d in reviewsPerUser[user]:
        i2 = d['beer/beerId']
        if i2 == item: continue
        ratings.append(d['review/overall'] - itemAverages[i2])
```

```
            if str(i2) in model10.wv:
                similarities.append(model10.wv.distance(str(item), str(i2)))
        if (sum(similarities) > 0):
            weightedRatings = [(x*y) for x,y in zip(ratings,similarities)]
            return itemAverages[item] + sum(weightedRatings) / sum(similarities)
        else:
            return ratingMean
```

In [71]:

```
u,i = reviewDicts[0]['user/profileName'],reviewDicts[0]['beer/beerId']
```

In [72]:

```
predictRating(u,i)
```

Out[72]:

```
3.88871
```

In [73]:

```
ratingMean
```

Out[73]:

```
3.88871
```

In [74]:

```
alwaysPredictMean = [ratingMean for _ in reviewDicts]
```

In [75]:

```
labels = [d['review/overall'] for d in reviewDicts]
```

In [76]:

```
predictions = [predictRating(d['user/profileName'],d['beer/beerId']) for d in re
```

In [77]:

```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

In [78]:

```
MSE(alwaysPredictMean, labels)
```

Out[78]:

```
0.4924295358999677
```

In [79]:

```
MSE(predictions, labels)
```

Out[79]:

```
0.4199912117748809
```

In [ ]: