**IST-664**

**Final Project Option 1**

**Pan Chen**

**Fall 2017**

**Prof. Nancy McCracken**

**Part 1: Simple sentiment classification**

**Preparation:**

I first imported the movie reviews sentences for classification, got all of the words from these sentences, and picked the most frequent 2000 words in that corpus to build the word\_features. Note that I did not shuffle the order of the review sentences in this step, on the contrary of what we did in the lab. I shuffled the featureset later for each cross-validation run.

**Baseline performances (unigram features):**

1. I defined the unigram features extraction function and generated unigram feature sets as per lab instructions.
2. I then shuffled the feature sets, and created the train sets and test sets from the shuffled feature sets. For the test sets I used the first 1000 shuffle feature sets, and for the train sets I used the rest of the shuffle sets (there are 9662 of them) to create a 90/10 split of our approximately 10,000 documents.
3. Then I trained the train sets with a Naïve Bayes classifier, and ran the 5-fold cross-validation to check the accuracies using the mean values of the 5 accuracies I obtained from running 5-fold CV.
4. Rerun the process from step 2 for two more times.

**The accuracies I got from 3 runs are:**

0 0.7514071294559099

1 0.75

2 0.7420262664165104

3 0.7434333958724203

4 0.7424953095684803

mean accuracy 0.7458724202626641

0 0.75093808630394

1 0.7415572232645403

2 0.75

3 0.7471857410881801

4 0.7415572232645403

mean accuracy 0.7462476547842402

0 0.7307692307692307

1 0.7406191369606003

2 0.7645403377110694

3 0.7528142589118199

4 0.7495309568480301

mean accuracy 0.7476547842401502

**Mean baseline accuracy:** **0.74659161976**

Let’s see if I can improve upon this with the definitions of some other features.

**Bigram feature combined with unigram feature**

1. I first created a bigram collocation finder using all of the original movie review words, and used the chi-squared measure to get bigrams that are informative features.
2. Then I created a feature extraction function that has bigram features and all of the unigram features from the previous section.
3. I generated feature sets using the feature function defined in the previous step
4. I then shuffled the feature sets, and created the train sets and test sets from the shuffled feature sets. For the test sets I used the first 1000 shuffle feature sets, and for the train sets I used the rest of the shuffle sets (there are XXX of them) to create a 90/10 split of our approximately 10,000 documents.
5. Then I trained the train sets with a Naïve Bayes classifier, and ran the 5-fold cross-validation to check the accuracies using the mean values of the 5 accuracies I obtained from running 5-fold CV.
6. Rerun the process from step 4 for two more times.

0 0.7467166979362101

1 0.74906191369606

2 0.7401500938086304

3 0.74906191369606

4 0.7560975609756098

mean accuracy 0.7482176360225141

0 0.7462476547842402

1 0.7406191369606003

2 0.7363977485928705

3 0.7410881801125704

4 0.7485928705440901

mean accuracy 0.7425891181988743

0 0.7293621013133208

1 0.7457786116322702

2 0.7626641651031895

3 0.7542213883677298

4 0.74812382739212

mean accuracy 0.748030018761726

**Mean bigram accuracy: 0.74627892432**

So, it seems that with the addition of bigram feature set, the predicting accuracy actually decreased by a ignorable fraction.

**POS feature combined with Bigram feature and unigram feature**

1. First, I created a feature extraction function that counts 4 types of POS tags nouns verbs, adjectives and adverbs as features. The function also included the bigram feature and unigram feature from the previous section.
2. Then I generated feature sets using the feature extraction function defined in the previous step.
3. I then shuffled the feature sets, and created the train sets and test sets from the shuffled feature sets. For the test sets I used the first 1000 shuffle feature sets, and for the train sets I used the rest of the shuffle sets (there are XXX of them) to create a 90/10 split of our approximately 10,000 documents.
4. Then I trained the train sets with a Naïve Bayes classifier, and ran the 5-fold cross-validation to check the accuracies using the mean values of the 5 accuracies I obtained from running 5-fold CV.
5. Rerun the process from step 2 for two more times.

0 0.7476547842401501

1 0.7420262664165104

2 0.7424953095684803

3 0.7471857410881801

4 0.724202626641651

mean accuracy 0.7407129455909944

0 0.7467166979362101

1 0.7476547842401501

2 0.7303001876172608

3 0.7424953095684803

4 0.74812382739212

mean accuracy 0.7430581613508442

0 0.7359287054409006

1 0.7514071294559099

2 0.7565666041275797

3 0.7387429643527205

4 0.7326454033771107

mean accuracy 0.7430581613508443

**Mean accuracy for POS+Bigram: 0.74227642276**

So it seems that with the addition of POS actually decreases the accuracy even more. So far it seems that the more feature sets I added in the worse the model predicted in the cross-validation. Let’s try another two more methods, separately.

**Linguistic Inquiry and Word Count Analyzer vs Sentiment Lexicon**

The additions of bigram features and POS features did not improve my predicting accuracy significantly, so next I would like to try to see if LIWC **(Linguistic Inquiry and Word Count) and sentiment lexicon can improve the predicting accuracy)**

I am not sure if I ran LIWC correctly so I just did it for fun…hope it would not affect my grade if I did something wrong.

**LIWC:**

1. Firstly I extracted a list of positive words and negative words from liwcdic2007.dic.
2. Then I created a feature extraction function that counts unique positive words and negative words of each sentence according to the positive and negative words list extracted from the previous step. If there are 2 unique positive words in a sentence, then its positive score is 2. The function also included the unigram feature from the baseline section.
3. Then I generated feature sets using the feature extraction function defined in the previous step.
4. I then shuffled the feature sets, and created the train sets and test sets from the shuffled feature sets. For the test sets I used the first 1000 shuffle feature sets, and for the train sets I used the rest of the shuffle sets (there are XXX of them) to create a 90/10 split of our approximately 10,000 documents.
5. Then I trained the train sets with a Naïve Bayes classifier, and ran the 5-fold cross-validation to check the accuracies using the mean values of the 5 accuracies I obtained from running 5-fold CV.
6. Rerun the process from step 2 for two more times.

Here are the CV results:

0 0.75

1 0.7560975609756098

2 0.7467166979362101

3 0.74812382739212

4 0.7340525328330206

mean accuracy 0.7469981238273921

0 0.7570356472795498

1 0.7448405253283302

2 0.7439024390243902

3 0.7392120075046904

4 0.7382739212007504

mean accuracy 0.7446529080675421

0 0.7387429643527205

1 0.7401500938086304

2 0.7593808630393997

3 0.7453095684803002

4 0.7467166979362101

mean accuracy 0.7460600375234522

Mean accuracy of LIWC featuresets: 0.7459036898061289

Again, the accuracy decreased from the baseline…

**Sentiment Lexicon**

1. Firstly, I defined a function that could look up a word in "subjclueslen1-HLTEMNLP05.tff" and returns its strength, posTag, isStemmed, polarity values. However, we are only going to look at strength and polarity.
2. Then I created a feature extraction function that counts unique words in each sentence’s strength and polarity. If the strength of the word is 'weaksubj' and its polarity is 'positive' then we add one to its weakPos score. If the strength of the word is 'strongsubj' and its polarity is ‘negative’ then we add one to its strongPos score. If the strength of the word is 'weaksubj' and its polarity is ‘negative’ then we add one to its weakNeg score. If the strength of the word is 'strongsubj' and its polarity is ‘negative’ then we add one to its StrongNeg score. The final Positive score for each sentence is weakPos + (2 \* strongPos), and the final Negative score to each sentence is weakNeg + (2 \* strongNeg). The function also included the unigram feature from the baseline section.
3. Then I generated feature sets using the feature extraction function defined in the previous step.
4. I then shuffled the feature sets, and created the train sets and test sets from the shuffled feature sets. For the test sets I used the first 1000 shuffle feature sets, and for the train sets I used the rest of the shuffle sets (there are XXX of them) to create a 90/10 split of our approximately 10,000 documents.
5. Then I trained the train sets with a Naïve Bayes classifier, and ran the 5-fold cross-validation to check the accuracies using the mean values of the 5 accuracies I obtained from running 5-fold CV.
6. Rerun the process from step 2 for two more times.

Here are the CV results:

0 0.7514071294559099

1 0.7640712945590994

2 0.7640712945590994

3 0.7696998123827392

4 0.7542213883677298

mean accuracy 0.7606941838649155

0 0.7514071294559099

1 0.7640712945590994

2 0.7640712945590994

3 0.7696998123827392

4 0.7542213883677298

mean accuracy 0.7606941838649155

0 0.7617260787992496

1 0.7584427767354597

2 0.7528142589118199

3 0.7668855534709194

4 0.7626641651031895

mean accuracy 0.7605065666041276

Mean accuracy of SL featuresets: 0.7459036898061289

To summarize: here are the accuracies of all the feature sets CV runs:

baseline=0.74659161976

bigram=0.74627892432

pos=0.74227642276

liwc=0.7459036898061289

sl=0.7607567229518448

The only featuresets that appears to increase the accuracy from the baseline performance is SL featuresets, while the others didn’t provide much difference no matter how they were combined.

**Part 2: Chatbot**

For this part of the project, I was trying to develop a chatbot that would understand what kind of food a customer wants to order and when does he/she wants it delivered for a restaurant. I developed two intents and 12 utterances to train it.

**Orderfood:**

The first intent I defined was “orderfood”, and I intended LUIS to capture three things (entities):

1. What kind of food a customer wants?
2. What’s the quantity of the food that he/she wants?
3. When does he/she want it?
4. Is the food he/she ordered for pick-up or delivery?

To address the first question, I defined a simple entity called “food”. I also defined a phrase list called “food too”, which collects the names of the food that people usually ordered for delivery, like “pepperoni pizza”, “pad thai”, and I’ve also added the values that were recommended by system: for example, when I added “pepperoni pizza” into my phrase list, it automatically suggested phrases like “Hawaiian pizza”, “deep-dish pizza” and so forth. My final “food” list.

To address the second question, I decided to use the prebuilt entity “number”.

To address the third question, I decided to use the prebuilt entity “datatimeV2”.

To address the fourth question, I defined a hierarchical entity called “method”, which contains two child entities “delivery” and “pickup”.

To train LUIS, I used 5 simple utterances for “orderfood” intent and labeled the appropriate entities to them:

1. i want a pepperoni pizza delivered at 10
2. can you deliver a dozen of wings to me at 10 : 30 tonight [?
3. can i order one pad thai for pick - up around noon ?
4. can i order two dozens of wings for pickup
5. can i get a box of noodles to - go in 10 minutes ?

**Trackfood:**

I’ve also defined a “trackfood” intent, as I intended LUIS could answer all the “where is my food” related questions from the customer after he/she placed an order.

I also used 6 simple utterances for “trackfood” intent and labeled the appropriate entities to them:

1. how long it would take you to get my pizza delivered ?
2. is my pizza ready for pickup ?is my food ready yet ?
3. when will my food arrive ?
4. where my food is ?
5. when will my pizza be delivered
6. is my food ready yet ?
7. when did the driver leave ?

Fortunately all of the entities defined in the “orderfood” intent are sufficient to cover all these questions.

I trained LUIS and published it to in eastern time zone:

<https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/bf14c774-af6f-4a05-b9ae-876c175ccb38?subscription-key=fc5fb906607c456db8ebacef5e48c17b&verbose=true&timezoneOffset=-300&q=>

One thing I noticed is that, LUIS does not seem to be able to recognize the word “a” as in the sense of describing the quantity of food a customer wants to order. For example, if a customer says “I want to order one pizza”, it is able to recognize the amount of pizza a customer wants to order is 1. However, if a customer says “I want to order a pizza”, LUIS does not seem to be able to see “a” as a number entity to describe the amount of pizza the customer wants to get.

Also it would be cool if it could respond to a customer’s request in “sessions”. For example, if a customer said “I want to order one chicken salad”, but 5 seconds later he said “sorry can I get two more?” It would be cool if LUIS can automatically update the number entity to 3.

Finally, the range of the language used in the customer-food delivery conversations are pretty limited in terms of vocabulary and grammar structure, and I am sure there will be more challenging situations when training LUIS for more complicated tasks like solving technical questions.