CP322 Project

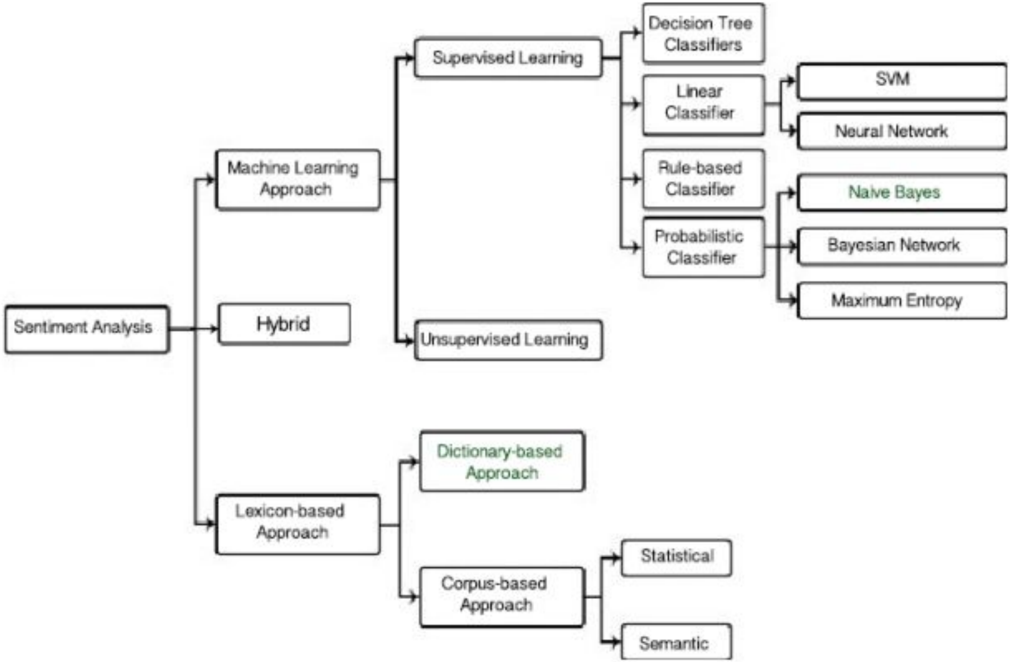
Group 9:
Phoebe Schulman, Anojan Balendra, Harchit Bhatoia,
Shing Hei Mau, Alvin Onabolu

	Clothing ID	Age	Title	Review Tex	Rating	Recommen	Positive Fe	Division Na	Departmer	Class Name
0	767	33		Absolutely	4	1	0	Initmates	Intimate	Intimates
1	1080	34		Love this di	5	1	4	General	Dresses	Dresses
2	1077	60	Some majo	I had such I	3	0	0	General	Dresses	Dresses
3	1049	50	My favorite	I love, love,	5	1	0	General Pe	Bottoms	Pants
4	847	47	Flattering s	This shirt is	5	1	6	General	Tops	Blouses
5	1080	49	Not for the	I love tracy	2	0	4	General	Dresses	Dresses
6	858	39	Cagrcoal sh	I aded this	5	1	1	General Pe	Tops	Knits
7	858	39	Shimmer, s	I ordered th	4	1	4	General Pe	Tops	Knits
8	1077	24	Flattering	I love this c	5	1	0	General	Dresses	Dresses
9	1077	34	Such a fun	I'm 5"5' an	5	1	0	General	Dresses	Dresses
10	1077	53	Dress looks	Dress runs	3	0	14	General	Dresses	Dresses

Introduction

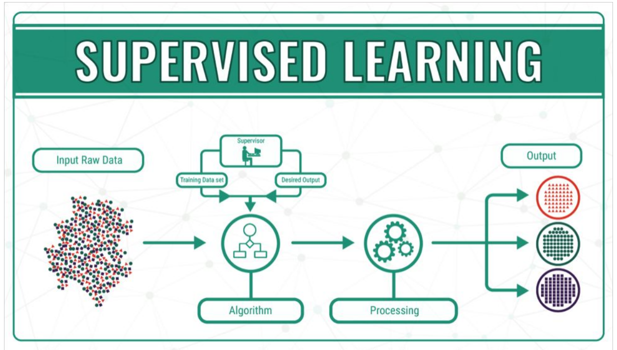
- Given data set of clothing reviews
- **Question being addressed**
  - How to develop a sentiment classifier that assigns reviews a rating from 1-5?
- Sentiment analysis
  - Analyzing text
  - Determine if an opinion is positive or negative

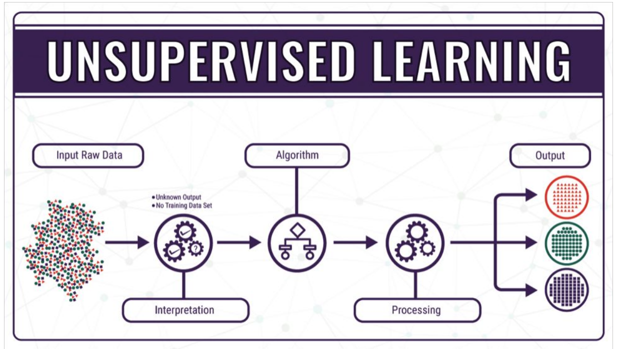
Related Work: Sentiment Analysis



Related Work: Machine Learning

- Supervised or Unsupervised?
- Naive Bayes
- K-Nearest Neighbors
- Support Vector Machine (SVM)
- Decision Trees





Naive Bayes + TFIDF

```
{'major': 1986,
 'design': 918,
 'flaw': 1275,
 'favorite': 1207,
 'buy': 493,
 'flattering': 1270,
 'shirt': 2831,
 'petite': 2364,
 'cagrcoal': 504,
 'shimmer': 2824,
 'fun': 1365,
 'surprisingly': 3175,
 'go': 1428,
 'lot': 1924,
 'dress': 1000,
 'look': 1901,
 'like': 1867,
 'make': 1989,
 'cheap': 578,
```

```
bow = transformer.transform(df["Title"])
print(bow)
```

Word's Unique value

index

Occurrence frequency

```
(2, 918)	1
(2, 1275)	1
(2, 1986)	1
(3, 493)	1
(3, 1207)	1
(4, 1270)	1
(4, 2831)	1
(5, 2364)	1
(6, 504)	1
(6, 1365)	1
(6, 2824)	1
(7, 1428)	1
(7, 1924)	1
(7, 2824)	1
(7, 3175)	1
(8, 1270)	1
(9, 1000)	1
(9, 1365)	1
(10, 578)	1
```

TFIDF importance score

```
(2, 1986)	0.7052716454501263
(2, 1275)	0.5691706469378793
(2, 918)	0.42265432777671547
(3, 1207)	0.6725653380818224
(3, 493)	0.7400377463419581
(4, 2831)	0.6619290578025505
(4, 1270)	0.7495664896702813
(5, 2364)	1.0
(6, 2824)	0.616833907181881
(6, 1365)	0.3567917478968672
(6, 504)	0.7015807719596033
(7, 3175)	0.4977749905370736
(7, 2824)	0.5923078260116152
(7, 1924)	0.479087681657613
(7, 1428)	0.41456783681943293
(8, 1270)	1.0
(9, 1365)	0.8220129508290713
```

```
title_tfidf = tfidf_trans.transform(bow)
print(title_tfidf)
```

Text data vs title data

```
array([[    0,     0,    22,    23,   797],
       [    0,     3,    29,    39,  1494],
       [    0,     0,    53,    40,  2778],
       [    0,     0,     1,   148,  4928],
       [    0,     0,     0,     4, 13127]], dtype=int64)
```

```
array([[   35,    61,   243,    89,   414],
       [    7,   151,   439,   262,   706],
       [    0,    25,   928,   499,  1419],
       [    0,     3,   128,  1179,  3767],
       [    0,     2,    28,   207, 12894]], dtype=int64)
```

Overall Accuracy: 56.76%

Class	n (truth) ?	n (classified) ?	Accuracy	Precision	Recall	F1 Score
1	0	842	96.41%	0.0	0.0	0.0
2	3	1565	93.35%	0.0019	1.0	0.0038
3	105	2871	87.78%	0.018	0.50	0.036
4	254	5077	78.56%	0.029	0.58	0.056
5	23124	13131	57.42%	1.0	0.57	0.72

Overall Accuracy: 64.66%

Class	n (truth) ?	n (classified) ?	Accuracy	Precision	Recall	F1 Score
1	42	842	96.53%	0.042	0.83	0.079
2	242	1565	93.59%	0.096	0.62	0.17
3	1766	2871	88.16%	0.32	0.53	0.40
4	2236	5077	78.9%	0.23	0.53	0.32
5	19200	13131	72.14%	0.98	0.67	0.80

Solution/Method

What was done

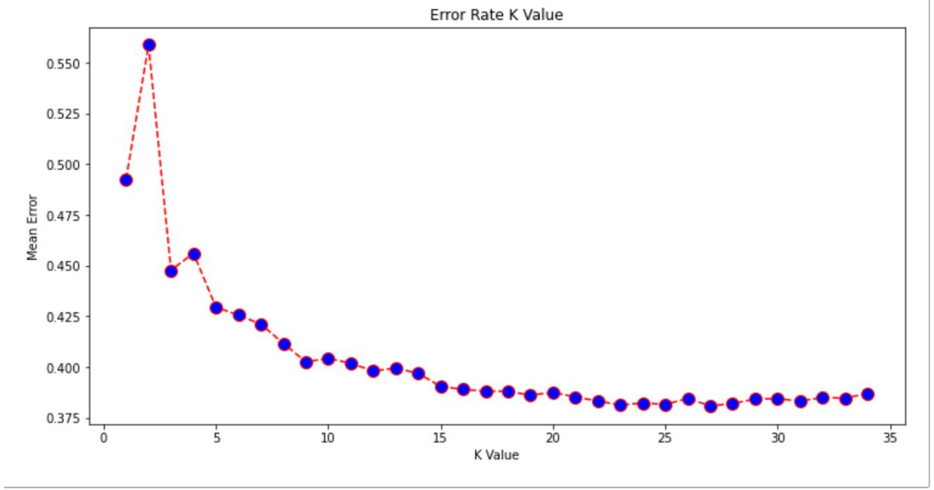
- Sentiment classification during preprocessing
- Predictive classifier: K-Nearest Neighbors
  - Why: better results than Gaussian Naive Bayes
  - Where K value changes
    - Why: best K value changed each time

```
#find a k value for knn- look for lowest point(error) on graph

minError = 1 #where is least erorr
minErrorK = 0 #which k to choose
errors = [0] #errors = []

# Calculate error for dif K values
for i in range(1,35):#(5,35):#(1, 40)

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(descriptive_train, target_train)
    pred_i = knn.predict(descriptive_test)
    errors.append(np.mean(pred_i != target_test))

    #print(errors[i-1],np.mean(pred_i != target_test),minError,minErrorK)
    if(errors[i]<minError): #if(errors[i-1]<minError):
        minError=errors[i]
        minErrorK=i

errors=errors[1:]
print("choose k = ", minErrorK)

plt.figure(figsize=(12, 6))
plt.plot(range(1, 35), errors, color='red', linestyle='dashed', marker='o', markerfacecolor='blue',

plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Solution/Method

Tools and techniques

- Finding best K value for KNN
  - Run many KNN
  - Calculate error for each K value
  - Find lowest point (least error) on graph
  - Use the corresponding K for actual predictions



choose k = 27

Solution/Method

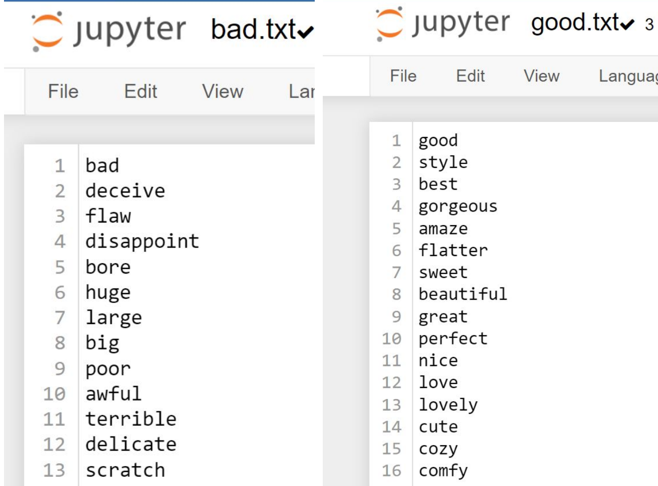
Tools and techniques

- Tokenization
    - Why: convert sentences into words
- Lemmatization
    - Reduces words to their root word
    - Eg: flattering -> flatter, flattered -> flatter
    - Why: not need to consider every version of a word

Data and Experiments

Preprocessing data

- Read files containing good or bad words
  - Added specific words from data
    - Eg: chic, cute, fab, itchy
- Sentiment analysis on Title and Review Text



Data and Experiments

Preprocessing data

- Evaluating Title and Review Text features
    - Replace nan with 0
    - Finding number of good/bad words in sentences
    - **final_eval = num_good_words - num_bad_words**
        - Positive when more good words
        - Negative when more bad words
    - Handle negation (“no”, “not”) -> switch the outcome
    - Handle expression (“!” , “too”) -> increase the outcome

```
#title
title_evals= []


for i in range(len(sentiment_df_title)):#for each sentence
    num_good_words = 0
    num_bad_words = 0

    final_eval = 0

    was_negated = False
    was_expression = False


    if (sentiment_df_title.values[i]!=0):#not na

        sentence = sentiment_df_title.values[i]
        print(sentence)#debug
        words=word_tokenize(sentence)

        for j in range(len(words)): #for each word

            word = words[j].lower()
            rootWord = lem.lemmatize(word,"v") #root words

            #meaning of word
            if rootWord in goodWords :
                num_good_words+=1

            elif rootWord  in badWords:
                num_bad_words+=1


            elif rootWord in negateWords:
                was_negated = True
            elif rootWord in expressionWords:
                was_expression = True
```

Data and Experiments

Examples

- Bad: Itchy
- Negated: Not impressed
  - impressed-> impress which is good
  - Not added -> bad
- Expression: Great shirt!!!
  - Great is positive
  - ! -> even more positive
- Combination: Not very flattering

```
Itchy tags
was_negated,was_expression =  False False
num_good_words, num_bad_words =  0 1
final_eval = -1
```

```
Not impressed...
was_negated, was_expression =  True False
num_good_words, num_bad_words =  1 0
final_eval = -1
```

```
Great shirt!!!
was_negated, was_expression =  False True
num_good_words, num_bad_words =  1 0
final_eval = 2
```

```
Not very flattering
was_negated, was_expression =  True True
num_good_words, num_bad_words =  1 0
final_eval = -2
```

Data and Experiments

```
descriptive_features===============

   Recommended IND  Positive Feedback Count  titleNew  textNew
0                1                        0         0        3
1                1                        4         0        8
2                0                        0        -1        6
3                1                        0         2      -14
4                1                        6         1        6
```

Preprocessing data

- Creating descriptive vs target features
  - Target features: Rating
  - Descriptive features: Title, Review Text, Positive Feedback Count, Recommended IND
    - Dropping remaining features
    - Replace Title and Review Text with final evaluations
  - Randomly split original data
    - 70% training : 30% testing

Data and Experiments

Reliable

- Robustness of good and bad words approach largely depends on the supply of good/bad words being checked against
- Larger word bank -> better results -> more reliable

Evaluation and Results

F1score: 0.555832

recall: 0.619075

Model evaluations

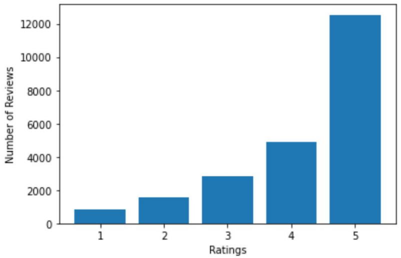
- F1 score: 55%
- Recall: 62%
- Others
    - F1 score for each rating (1-5)
    - Accuracy: 62%

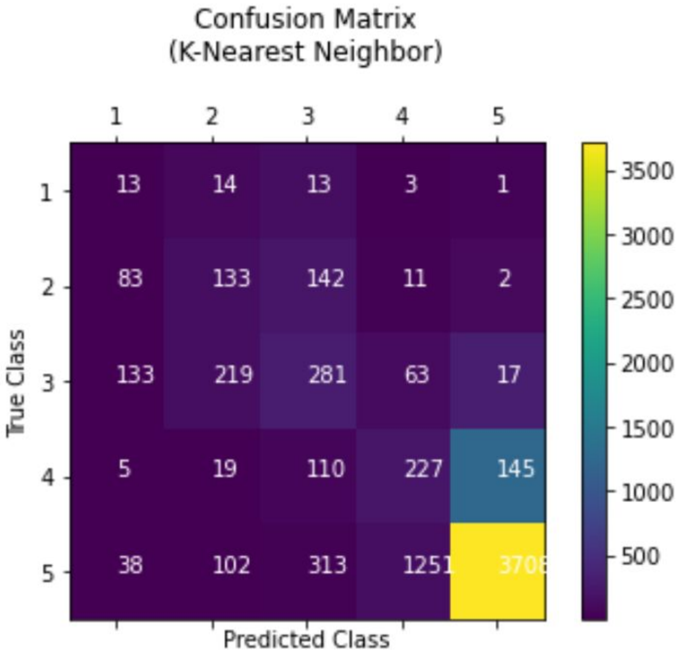
	precision	recall	f1-score	support
1	0.05	0.30	0.08	44
2	0.27	0.36	0.31	371
3	0.33	0.39	0.36	713
4	0.15	0.45	0.22	506
5	0.96	0.69	0.80	5412
accuracy			0.62	7046
macro avg	0.35	0.44	0.35	7046
weighted avg	0.79	0.62	0.68	7046

Evaluation and Results

Model evaluations

- Confusion matrix shows final ratings
- Eg:
  - 1,1 -> true positive of rating 1 (13)
  - 5,5 -> true positive of rating 5 (3708)
    - Why: imbalance data





Evaluation and Results

Conclusions

- Best results seen in KNN approach
- More robustness with manual preprocessing of data

```
'''
IN CONCLUSIONS...we get results similar to :

F1score: 0.555832
precision: 0.559843
recall: 0.619075

                  precision    recall  f1-score   support

           1       0.05      0.30      0.08        44
           2       0.27      0.36      0.31       371
           3       0.33      0.39      0.36       713
           4       0.15      0.45      0.22       506
           5       0.96      0.69      0.80      5412

    accuracy                           0.62      7046
   macro avg       0.35      0.44      0.35      7046
weighted avg       0.79      0.62      0.68      7046


therefore our model is about 62% accurate, F1score is 56%, and recall is 62% .

In addition:
f1 score of each target is different.
eg:
rating 1 with fscore= 0.08
vs
rating 5 with fscore= 0.80

this is either because the data is highly imbalanced or because we are better at predicting the 5 star ratings.
'''
```

Q and A