Yelp Review Ratings Prediction

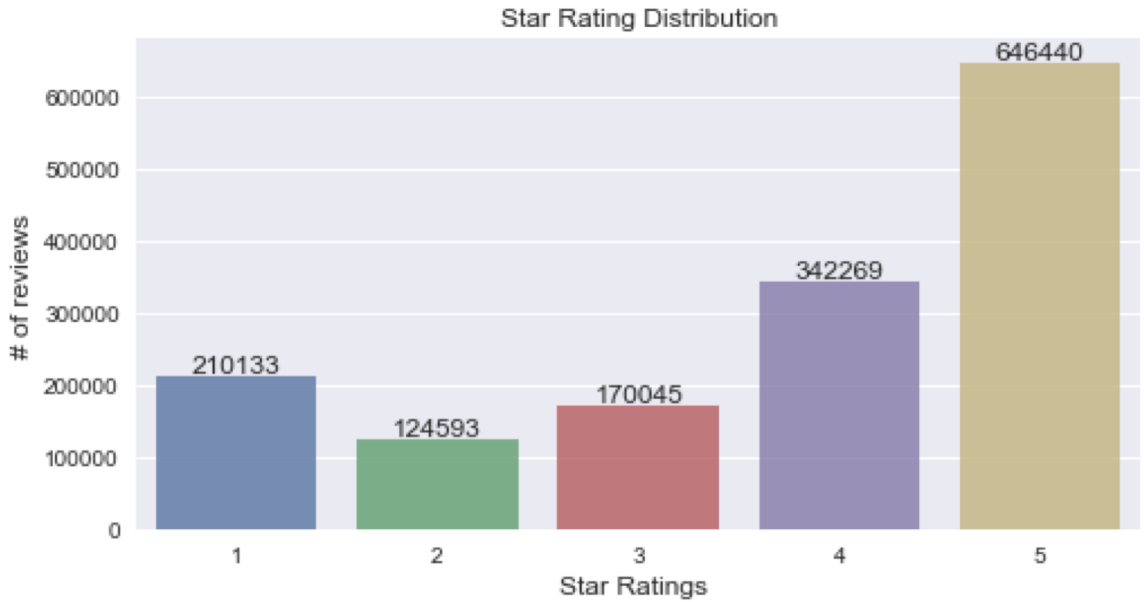
Math189 Final Project
Ziyuan Shang
Shihao Lin

Problem & Data Description

- What's in a review? Is it positive or negative?

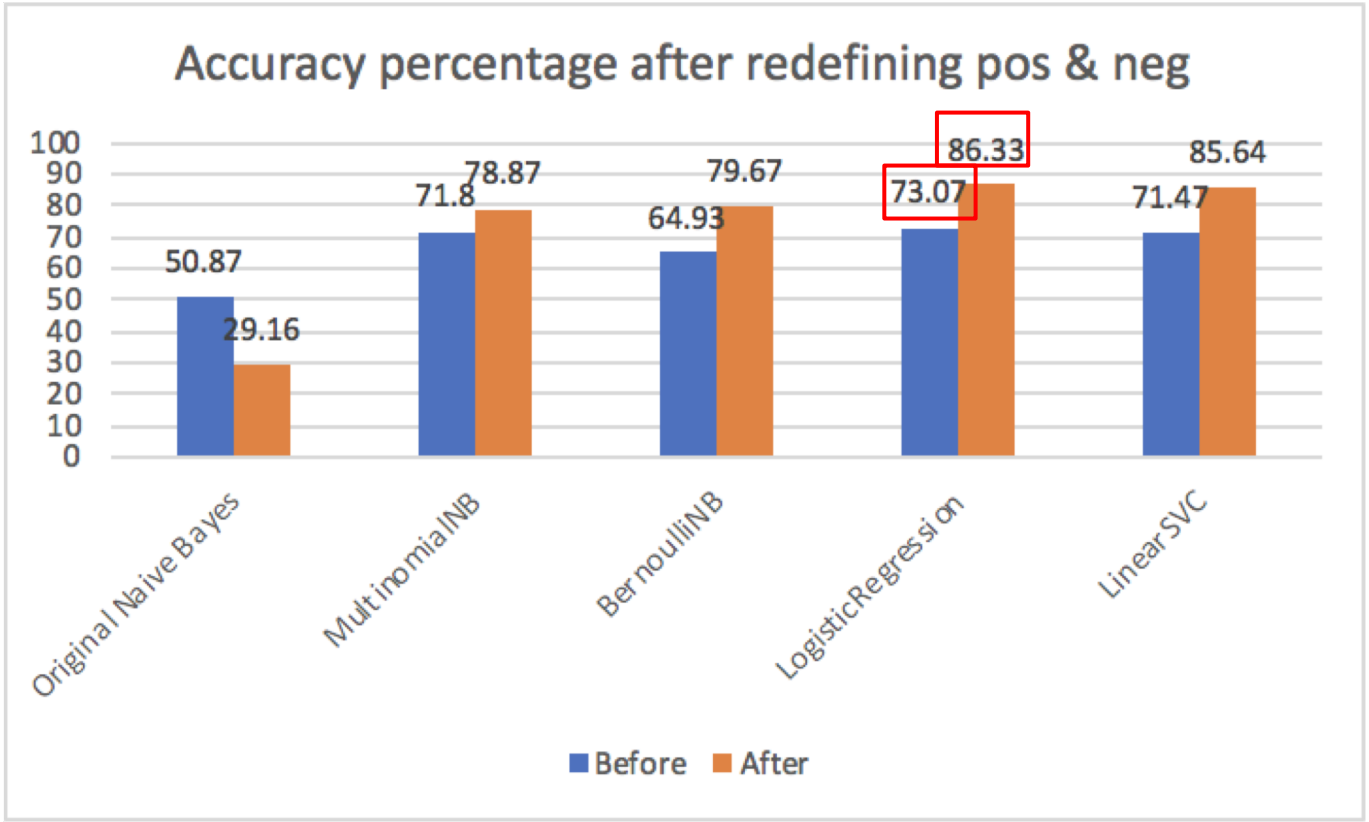
- **Review.json** {"text","stars"}

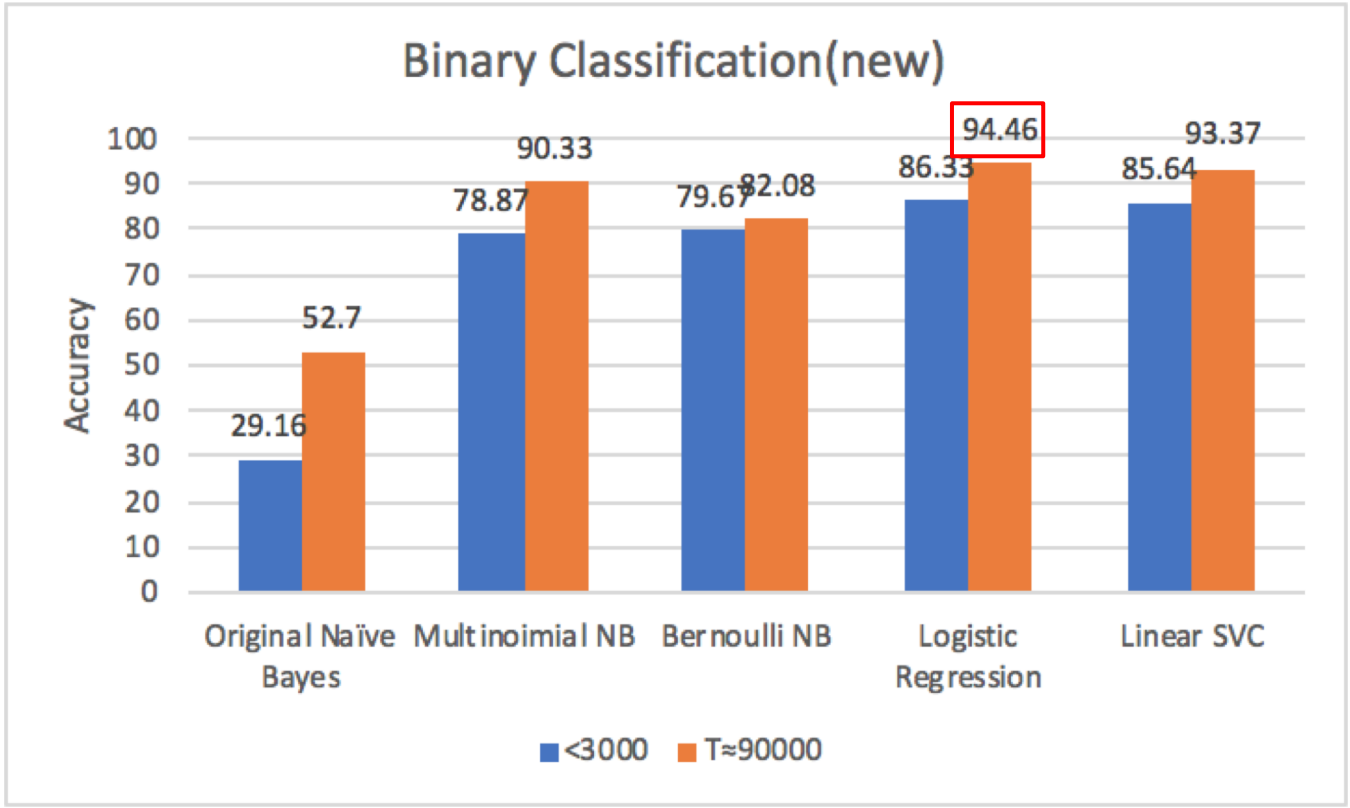
Review Star Distribution



Pos & Neg Definition

3000


Accuracy percentage after redefining pos & neg
100
90
80
70
60
50
40
30
20
10
0
50.87
29.16
71.8
78.87
64.93
79.67
73.07
86.33
71.47
85.64
Original Naive Bayes
MultinomialNB
BernoulliNB
LogisticRegression
LinearSVC
Before
After


Binary Classification(new)
Accuracy
100
90
80
70
60
50
40
30
20
10
0
29.16
52.7
78.87
90.33
79.67
82.08
86.33
94.46
85.64
93.37
Original Naïve Bayes
Multinoimial NB
Bernoulli NB
Logistic Regression
Linear SVC
<3000
T≈90000

Text Vectorization Scikit-learn Library

- **Word Counts with CountVectorizer**
- **Word Frequencies with TfidfVectorizer**
  - **Term Frequency**: This summarizes how often a given word appears within a document.
  - **Inverse Document Frequency**: This downscales words that appear a lot across documents.
- **Bigram (n-gram model)**

CountVectorizer

```
Smallest Coefs:
['preselected' 'negitive' 'excavated' 'phs' 'wrung' 'exodus' 'miscues'
 'montmartre' 'portishead' 'crimini' 'sprechers' 'conglomerates'
 'stinginess' 'sunridge' 'infuriated']

Biggest Coefs:
['jav' 'paddies' 'wihtout' 'dickinson' 'dobra' 'wizards' 'ssssooo' 'vt'
 'koval' 'vcr' 'bosas' 'robotics' 'preconceived' 'abusers' 'seagull']
```

Text Vectorization Scikit-learn Library

- Word Counts with CountVectorizer
- **Word Frequencies with TfidfVectorizer**
  - **Term Frequency**: This summarizes how often a given word appears within a document.
  - **Inverse Document Frequency**: This downscales words that appear a lot across documents.
- **Bigram (n-gram model)**

TfidfVectorizer

```
Smallest Coefs:
['worst' 'mediocre' 'disappointing' 'unprofessional' 'bland' 'horrible'
 'meh' 'downhill' 'tasteless' 'rude' 'terrible' 'lacked' 'awful' 'inedible'
 'poisoning' 'flavorless' 'unacceptable' 'worse' 'overpriced' 'poor'
 'disappointment' 'poorly' 'rudest' 'underwhelming' 'ok']

Biggest Coefs:
['amazing' 'delicious' 'great' 'excellent' 'awesome' 'best' 'fantastic'
 'perfect' 'perfection' 'highly' 'pleasantly' 'incredible' 'phenomenal'
 'love' 'perfectly' 'hesitate' 'notch' 'wonderful' 'thank' 'gem' 'grateful'
 'heaven' 'deliciousness' 'outstanding' 'superb']
```

Text Vectorization Scikit-learn Library

- **Word Counts with CountVectorizer**
- **Word Frequencies with TfidfVectorizer**
  - **Term Frequency**: This summarizes how often a given word appears within a document.
  - **Inverse Document Frequency**: This downscales words that appear a lot across documents.
- **Bigram (n-gram model)**

bigrams

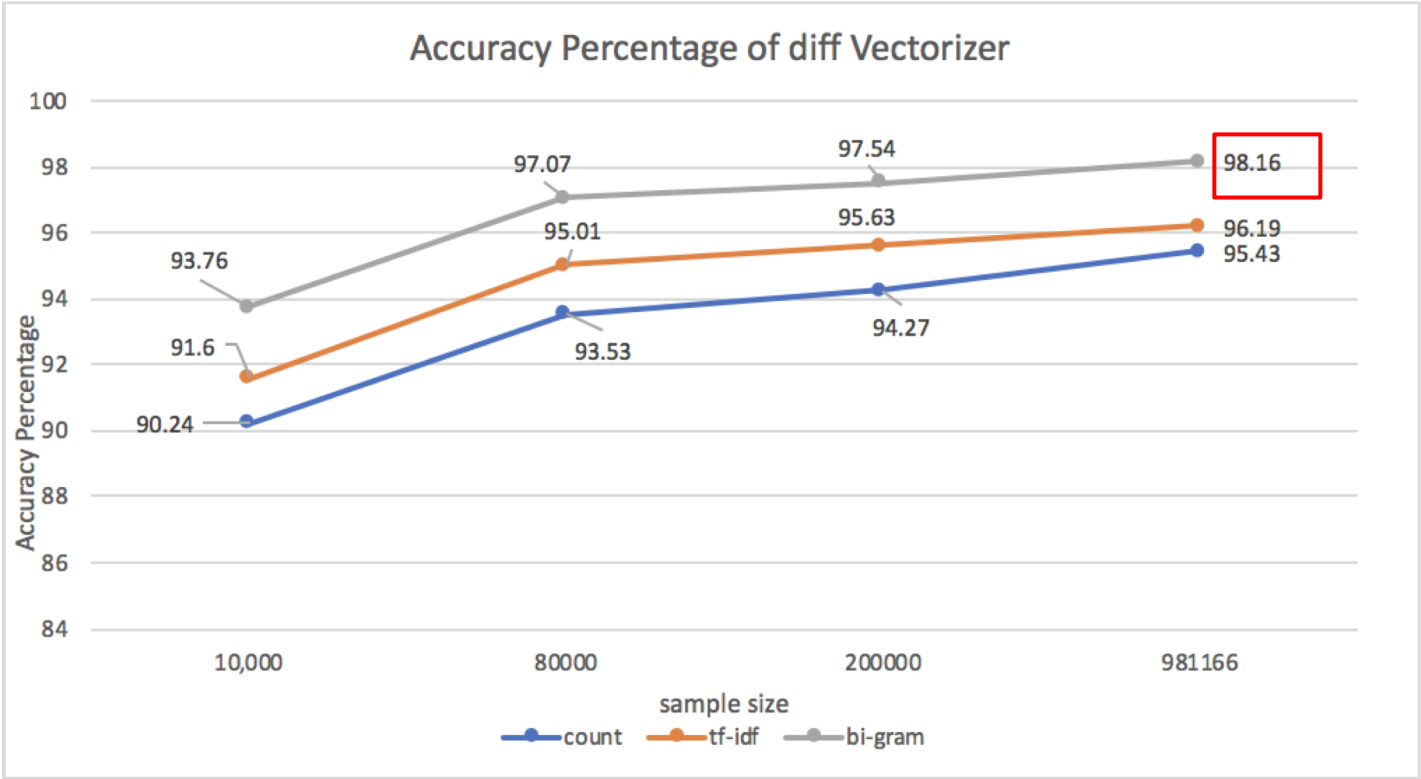
```
Smallest Coefs:
['worst' 'not worth' 'two stars' 'disappointing' 'horrible' 'rude' 'not'
 'terrible' 'bland' 'mediocre' 'at best' 'meh' 'not impressed' 'awful'
 'poor' 'worse' 'overpriced' 'disappointment' 'disappointed'
 'unprofessional' 'very disappointed' 'not good' 'tasteless' 'not very'
 'not great' 'unfortunately' 'never again' 'not recommend' 'to love'
 'disgusting' 'slow' 'lacked' 'nice but' 'however' 'poorly' 'average'
 'definitely not' 'over priced' 'that great' 'elsewhere' 'lacking'
 'wont be' 'dirty' 'nothing' 'one star']
```

bigrams

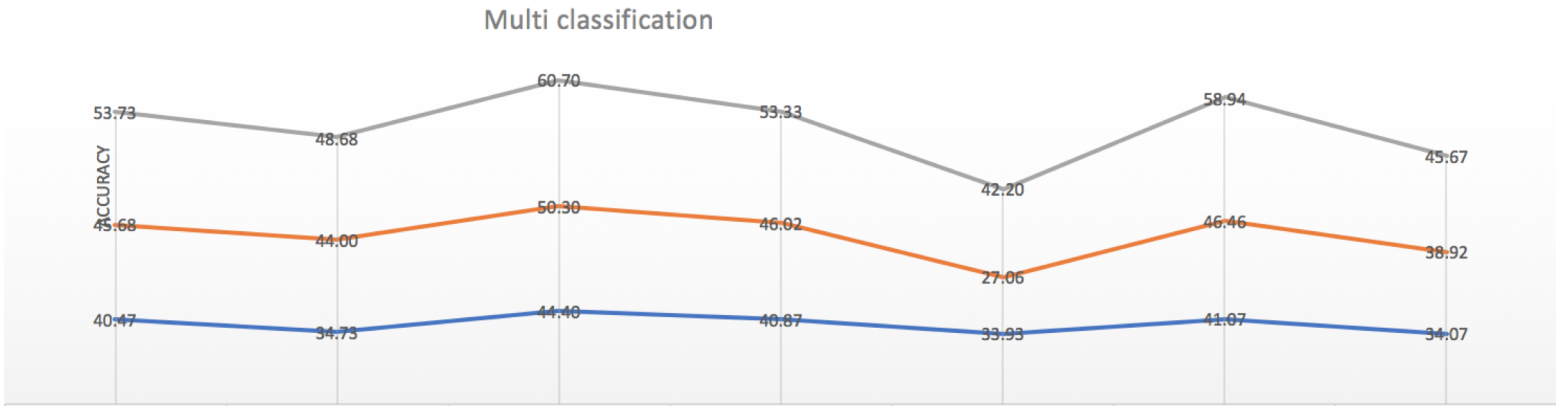
```
Biggest Coefs:
['amazing' 'great' 'delicious' 'best' 'excellent' 'awesome'
 'be disappointed' 'fantastic' 'perfect' 'not disappointed' 'love'
 'wonderful' 'incredible' 'outstanding' 'not only' 'never disappointed'
 'highly recommend' 'loved' 'you wont' 'friendly' 'perfection' 'definitely'
 'been disappointed' 'perfectly' 'even better' 'phenomenal'
 'will definitely' 'happy' 'love this' 'so good' 'five stars' 'heaven'
 'favorite' 'thank you' 'honest' 'thanks' 'no charge' 'definitely be'
 'yummy' 'good' 'superb' 'professional' 'on point' 'better than' 'thank']
```

LinearSVC

98.16%



Multi-classification on stars



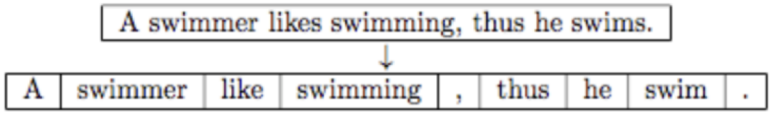
	MNB_classifier:	BernoulliNB_classifier:	LogisticRegression:	SGDClassifier_dassifier:	SVC_dassifier:	LinearSVC_classifier:	RandomForest_classifier:
50000	(25000 training:25000 testing	53.73	48.68	60.70	53.33	42.20	58.94	45.67
10000	(5000 training:5000 testing)	45.68	44.00	50.30	46.02	27.06	46.46	38.92
3000	(1500 training:1500 testing)	40.47	34.73	44.40	40.87	33.93	41.07	34.07

LDA(latent Dirichlet allocation)

- Implementation: Python’s Gensim package, spacy (lemmatization)
- Visualization: pyLDAvis

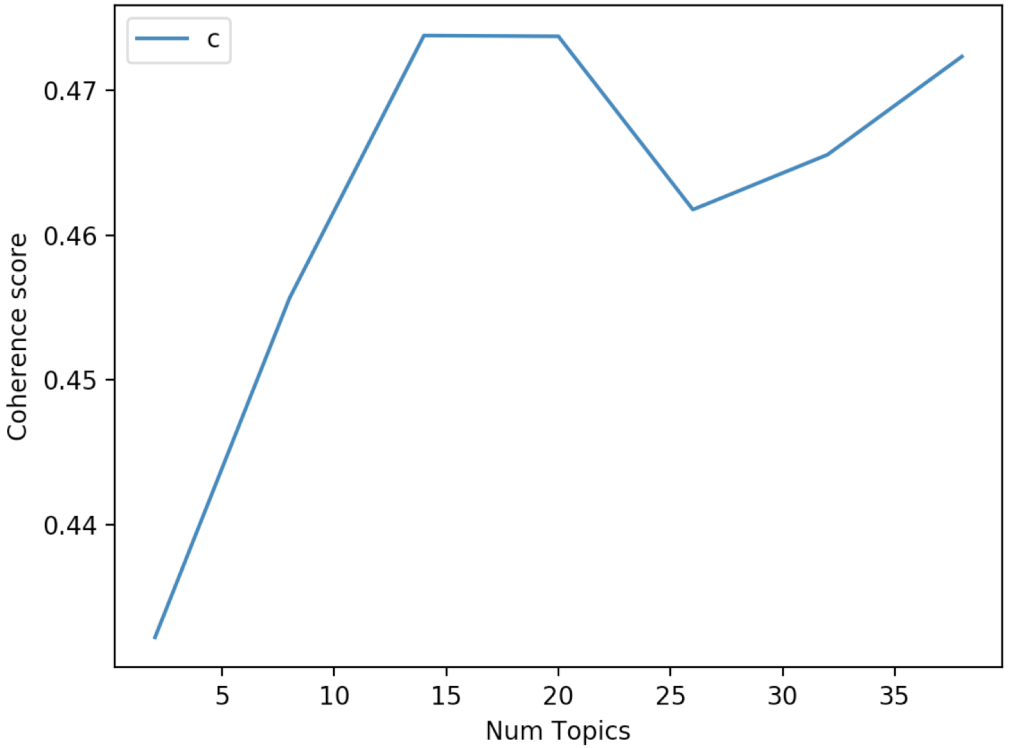
Text Processing

- Word tokenization and clean-up text
  - tokenize each sentence into a list of words, removing punctuations and unnecessary characters
  - Gensim's `simple_preprocess()`
- Remove stop words
- Make bigrams
  - Gensim's `Phrases` model
- lemmatization
- Create dictionary and corpus needed for Topic modeling



How to select k?

Num Topics = 2 has Coherence Value of 0.4322
Num Topics = 8 has Coherence Value of 0.4556
Num Topics = 14 has Coherence Value of 0.4738
Num Topics = 20 has Coherence Value of 0.4738
Num Topics = 26 has Coherence Value of 0.4618
Num Topics = 32 has Coherence Value of 0.4656
Num Topics = 38 has Coherence Value of 0.4724



Topics in LDA model

```
[(10,
  '0.075*"customer" + 0.074*"service" + 0.041*"bad" + 0.035*"manager" + '
  '0.022*"pay" + 0.021*"employee" + 0.019*"speak" + 0.018*"receive" + '
  '0.018*"rude" + 0.014*"wrong"'),
  (4,
  '0.064*"chicken" + 0.034*"dish" + 0.032*"sauce" + 0.024*"beef" + '
  '0.023*"taste" + 0.022*"spicy" + 0.021*"order" + 0.020*"rice" + '
  '0.017*"noodle" + 0.016*"flavor"'),
...]
```

Visualization using pyLDAvis

Binary Classification (LinearSVC)

98.16%