CS 410 Project – Understanding the Sentiment of Restaurant Reviews Based on Text Content

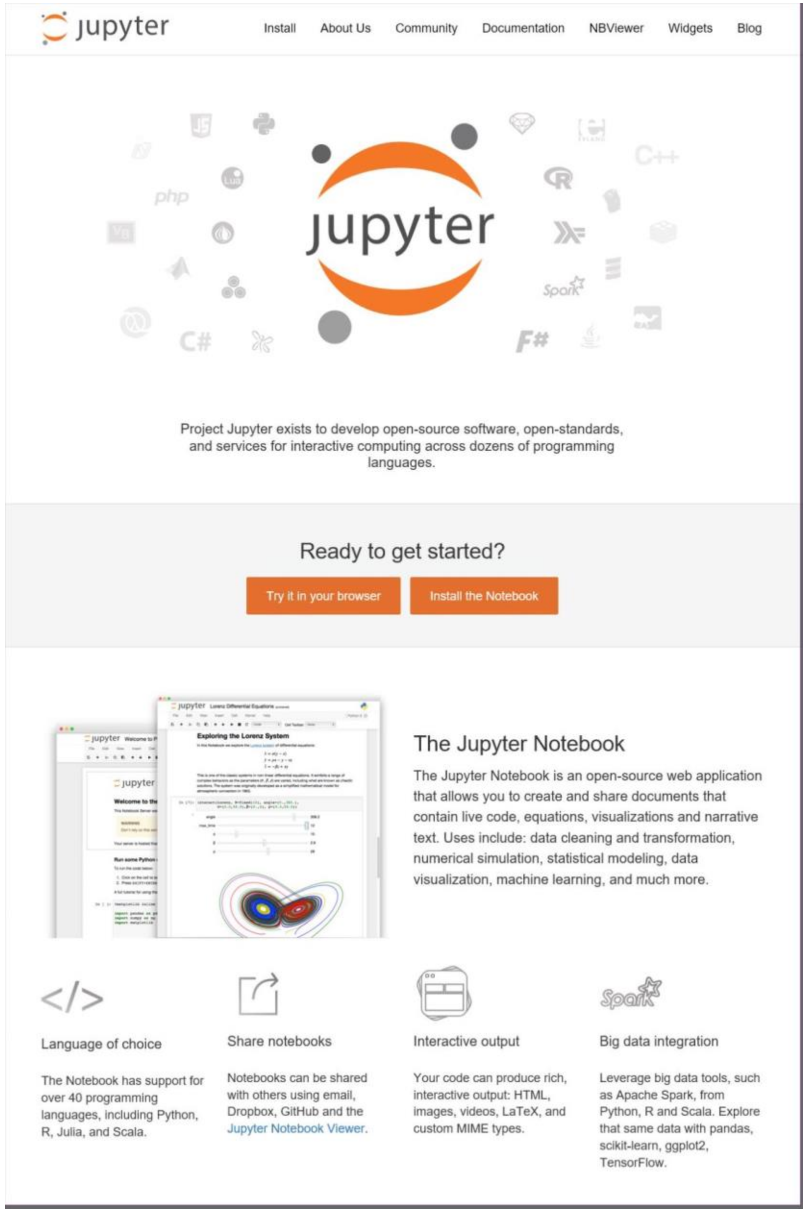
Hooman Shirani-Mehr

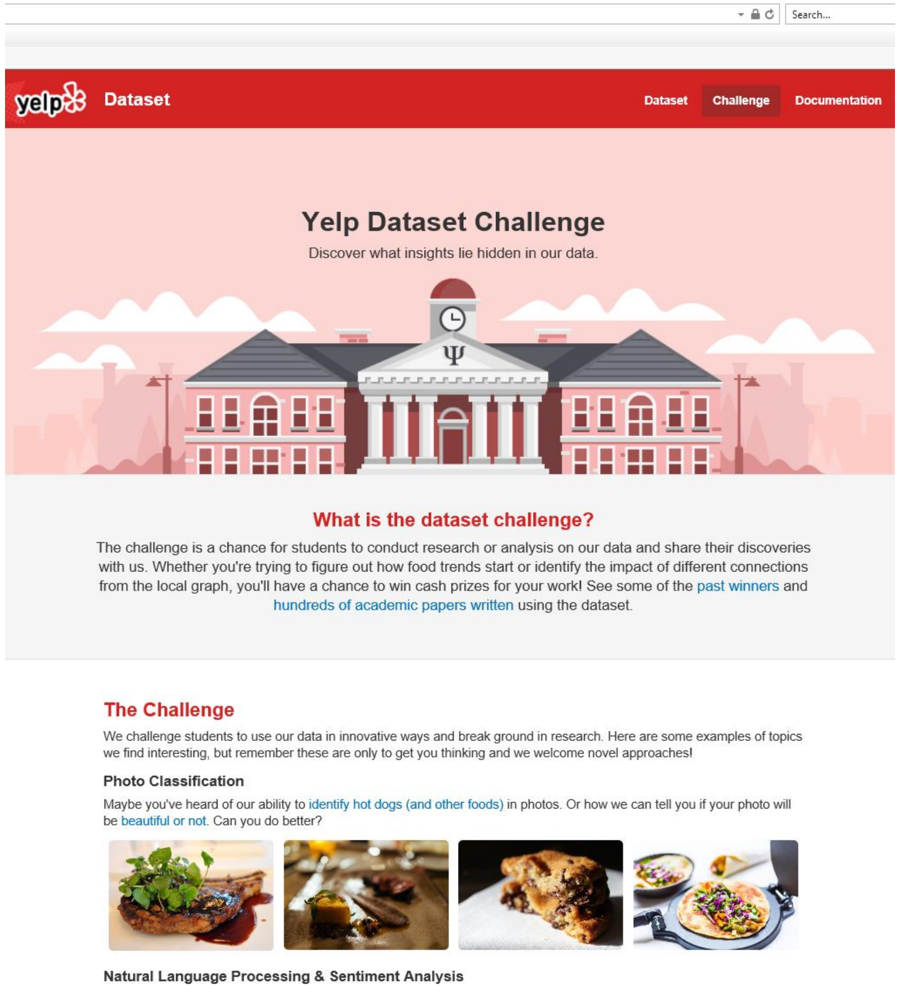
Dec 21, 2017

Overview

- Jupyter notebook is used for this project
  https://jupyter.org/

- Yelp dataset is used https://www.yelp.com/dataset/challenge





Function – Reading Data from File

- Function "readCvsData" is used to read data from the data file.
- Function inputs:
  - "fileName" which is the name of the data file in the local directory.
  - "numComments" which indicates the number of comments to be read from the file.
- File is opened as CVS data file and "numComments" comments are read using a for loop.
- Function outputs:
  - "scores" is the set of ratings read from the data file.
  - "comments" is the set of comments read from the data file.

```python
def readCvsData(fileName, numComments):

    # Initialize variables
    scores = []
    comments = []
    idx = 0

    with open(fileName) as csvDataFile:

        csvReader = csv.reader(csvDataFile)

        # Loop over row
        for row in csvReader:

            idx = idx+1

            if 0 < idx < numComments+1:

                scores.append(row[0])
                comments.append(row[1])

    # Return scores and comments read from the input CVS file
    return scores, comments
```

Function – Generating Word Cloud

- Function "generateWordCloud" is used to illustrate words in comments based on their frequency i.e. the more word appears, the bigger it is in the word cloud.
- Function inputs:
  - "wordData" is the collection of words in the comments.
- Function "WordCloud" is used to generate the word cloud. The result is then plotted and shown on the screen.
- Function outputs:
  - None

```python
def generateWordCloud(wordData):

    # Generate word cloud
    wordcloud = WordCloud(stopwords=STOPWORDS,
                          background_color='white',
                          width=1200,
                          height=1000
                         ).generate(wordData)

    # Plot the result
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.show()

    return
```

Function – Generating Pie Chart

- Function "generatePieChart" is used to illustrate distribution of ratings/scores in the data i.e. it shows what percentage of ratings are 1 star, what percentage are 2 stars and so on.
- Function inputs:
  - "scoreData" is the collections of ratings/scores.
- Score data is processed and formed into a "data frame". The result is used to plot the pie chart.
- Function outputs:
  - None

```
def generatePieChart(scoreData):

    # Form data frame using rating/score and number or occurance of each score
    values = [scoreData.count("1"), scoreData.count("2"),
              scoreData.count("3"), scoreData.count("4"),
              scoreData.count("5")]
    rawData = {'Rating' : ['1 star', '2 stars', '3 stars',
                           '4 stars', '5 stars'],
               'Count' : values}
    df = pd.DataFrame(rawData, columns = ['Rating', 'Count'])

    # Plot pie chart
    plt.figure(figsize=(16,8))

    df.plot(kind='pie', subplots=True, autopct='%1.1f%%',
            startangle=90, shadow=False, labels=df['Rating'],
            legend = False, fontsize=14)
    plt.show()

    return
```

Function – Balancing Classes

- Function "balanceClasses" is used to balance data For example if in our training data set we have 100 1-star reviews, 200 2-star reviews, 300 3-star reviews, 400 4-star reviews and 500 5-star reviews, only 100 reviews in each class will be considered.
- Function inputs:
  - "initComments" is the initial (unbalanced) set of comments.
  - "initScores" is the initial (unbalanced) set of scores.
- For each class (1-start, 2-star, ..., 5-star), frequency is calculated. The minimum number is considered as maximum allowable size of each class.
- Function outputs:
  - "newComments" is the new set of comments that are balanced.
  - "newScores" is the new set of scores that are balanced.

```
from collections import Counter

def balanceClasses(initComments, initScores):

    # Count number of occurance for each score
    freqScores = Counter(initScores)

    # Consider the least frequent class as the maximum number
    maxAllowable = freqScores.most_common()[-1][1]

    # Initialize variables
    numAdded = {clss: 0 for clss in freqScores.keys()}
    newComments = []
    newScores = []

    # Only consider maxAllowable number of comments and scores for each class
    for i, y in enumerate(initScores):
        if numAdded[y] < maxAllowable:
            newComments.append(initComments[i])
            newScores.append(y)
            numAdded[y] += 1

    # Return updated comments and scores
    return newComments, newScores
```

Function – Creating Document-Term Matrix (DTM)

- Function "createDTM" creates a document-term matrix for the data to describes the frequency of terms that occur in a collection of documents.
- Function inputs:
  - "messages" is the collection of all comments.
- For each word TF-IDF is calculated first i.e., the elements in DTM are calculated as the product of two weights, the term frequency and the inverse document frequency. The lower and upper boundary of the range of n-values for different n-grams are considered to be 1 and 2.
- Function outputs:
  - "dtm" is the calculated DTM.

```
from sklearn.feature_extraction.text import TfidfVectorizer

def createDTM(messages):

    # Calculate TF-IDF value for each word using 1-gram and 2-gram models
    vect = TfidfVectorizer(ngram_range=(1,2))

    # Create DTM
    dtm = vect.fit_transform(messages)

    # Return DTM
    return dtm
```

Function – Linear Support Vector Classification

- Function "LinSvcPrediction" uses Linear Support Vector Classification for predicting scores.
- Function inputs:
  - "xTrain" is the set of features for training.
  - "yTrain" is the set of scores for training.
  - "xTest" is the set of features for testing.
- The function first initializes the classifier, then trains the classifier using features and scores in training data set, and finally predicts scores for features in the test set.
- Function outputs:
  - "scorePred" is the set of scores/ratings predicted based on training data and by using linear support vector classification.

```python
from sklearn.svm import LinearSVC

def LinSvcPrediction(xTrain, yTrain, xTest):

    # Initialize the SVM classifier
    model = LinearSVC(penalty="l2", loss="squared_hinge", dual=True, tol=0.0001,
                      C=0.2, multi_class="ovr", fit_intercept=True, intercept_scaling=1,
                      class_weight=None, verbose=0, random_state=None, max_iter=1000)

    # Train the classifier using training data
    model.fit(xTrain, yTrain)

    # Predict scores
    scorePred = model.predict(xTest)

    # Return predicted scores
    return scorePred
```

Function – Random Forest Classification

- Function "randomForestClassification" uses random forest classification for predicting scores.
- Function inputs:
  - "xTrain" is the set of features for training.
  - "yTrain" is the set of scores for training.
  - "xTest" is the set of features for testing.
- The function first initializes the classifier, then trains the classifier using features and scores in training data set, and finally predicts scores for features in the test set.
- Function outputs:
  - "scorePreds" is the set of scores/ratings predicted based on training data and by using random forest classification.

```python
from sklearn.ensemble import RandomForestClassifier

def randomForestClassification(xTrain, yTrain, xTest):

    # Initialize the classifier
    model = RandomForestClassifier(min_samples_leaf = 10,
                                   max_features = None, n_estimators = 10)

    # Train the classifier using training data
    model.fit(xTrain, yTrain)

    # Predict scores
    scorePred = model.predict(xTest)

    # Return predicted scores
    return scorePred
```

Function – Logistic Regression Classifier

- Function "logisticRegressionClassification" uses linear regression to predict scores.
- Function inputs:
  - "xTrain" is the set of features for training.
  - "yTrain" is the set of scores for training.
  - "xTest" is the set of features for testing.
- The function first initializes the classifier, then trains the classifier using features and scores in training data set, and finally predicts scores for features in the test set.
- Function outputs:
  - "scorePreds" is the set of scores/ratings predicted based on training data and by using linear regression.

```
from sklearn.linear_model import LogisticRegression

def logisticRegressionClassification(xTrain, yTrain, xTest):

    # Create a logistic regression classifier
    model = LogisticRegression(penalty="l2", dual=False, tol=0.0001,
                               C=1.5, fit_intercept=True, intercept_scaling=1,
                               class_weight=None, random_state=None,
                               solver="liblinear", max_iter=100, multi_class="ovr",
                               verbose=0, warm_start=False, n_jobs=1)

    # Train the model using the training sets
    model.fit(xTrain, yTrain)

    # Predict output
    scorePred = model.predict(xTest)

    # Return predicted scores
    return scorePred
```

Function – Multinomial Naive Bayes Classification

- Function "multinomialNaiveBayesClassification" uses multinomial Naive Bayes classifier to predict scores.
- Function inputs:
  - "xTrain" is the set of features for training.
  - "yTrain" is the set of scores for training.
  - "xTest" is the set of features for testing.
- The function first initializes the classifier, then trains the classifier using features and scores in training data set, and finally predicts scores for features in the test set.
- Function outputs:
  - "scorePreds" is the set of scores/ratings predicted based on training data and by using multinomial Naive Bayes classification.

```python
from sklearn.naive_bayes import MultinomialNB

def multinomialNaiveBayesClassification(xTrain, yTrain, xTest):

    # Create a multinomial Naive Bayes classifier
    model = MultinomialNB(alpha=0.1, fit_prior=True, class_prior=None)

    # Train the model using the training sets
    model.fit(xTrain, yTrain)

    # Predict output
    scorePred = model.predict(xTest)

    # Return predicted scores
    return scorePred
```

Main

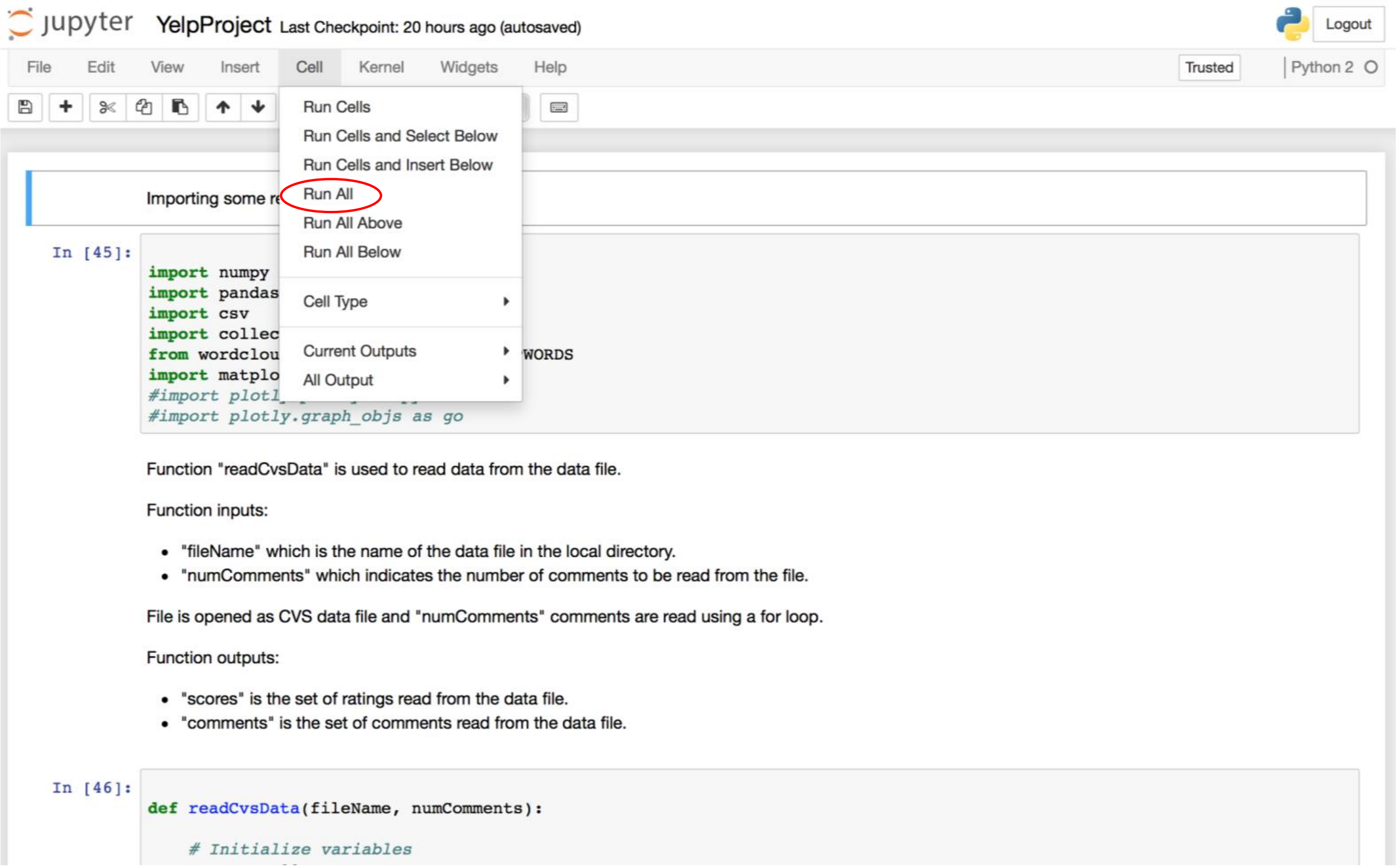
```python
def main():

    import warnings
    warnings.filterwarnings("ignore")

    # Load the csv file into pandas dataframe
    InputFile = 'data.csv'
    numDataEntriesToRead = 10000
    allScores, allComments = readCvsData(InputFile, numDataEntriesToRead)

    # Word cloud for 1 start and 5 star reviews
    outStr1 = ""
    outStr5 = ""
    idx = 0
    for xxx in allComments:
        if allScores[idx]=='1':
            outStr1 = outStr1 + ' ' + allComments[idx]
        if allScores[idx]=='5':
            outStr5 = outStr5 + ' ' + allComments[idx]
        idx = idx+1
    generateWordCloud(outStr1)
    generateWordCloud(outStr5)

    # Generate pie chart
    generatePieChart(allScores)

    # Balance classes to have same number of reviews for all classes (1-star, ..., 5-star)
    print("Number of comments in each class before balancing: ", Counter(allScores))
    balancedComments, balancedScores = balanceClasses(allComments, allScores)
    print("Number of comments in each class before balancing: ", Counter(balancedScores))
    allScores = balancedScores
    allComments = balancedComments

    # Extract features
    allFeatures = createDTM(allComments)

    allScores = pd.DataFrame(allScores)
```

```python
    from sklearn.model_selection import KFold
    kf = KFold(n_splits=10)

    # Initialize accuracies for all classification algorithm
    accuracyMxSvc = np.zeros(10)
    accuracyMxRfc = np.zeros(10)
    accuracyMxLr  = np.zeros(10)
    accuracyMxMnb = np.zeros(10)

    # Initialize the counter for K-fold cross validation
    idx = 0

    KFold(n_splits=10, random_state=None, shuffle=False)

    for train_index, test_index in kf.split(allComments):

        # Partition data into trainng data and test data
        featureTrain, featureTest = allFeatures[train_index,:], allFeatures[test_index,:]
        scoreTrain, scoreTest = allScores.loc[train_index], allScores.loc[test_index]

        # Predict using linear SVC classifier
        scoreSvcPred = LinSvcPrediction(featureTrain, scoreTrain, featureTest)

        # Predict using random forest
        scoreRfcPred = randomForestClassification(featureTrain, scoreTrain, featureTest)

        # Predict using logistic regression
        scoreLrPred = logisticRegressionClassification(featureTrain, scoreTrain, featureTest)

        # Predict using multinomial Naive Bayes
        scoreMnbPred = multinomialNaiveBayesClassification(featureTrain, scoreTrain, featureTest)

        # Calculate accuracy score for all algorithms
        from sklearn.metrics import accuracy_score
        accuracyMxSvc[idx] = accuracy_score(scoreTest, scoreSvcPred)
        accuracyMxRfc[idx] = accuracy_score(scoreTest, scoreRfcPred)
        accuracyMxLr[idx]  = accuracy_score(scoreTest, scoreLrPred)
        accuracyMxMnb[idx] = accuracy_score(scoreTest, scoreMnbPred)

        # Display accuracy scores
        print("idx: ", idx, " Accuracy Score SVC: ", accuracyMxSvc[idx])
        print("idx: ", idx, " Accuracy Score RFC: ", accuracyMxRfc[idx])
        print("idx: ", idx, " Accuracy Score LR: ", accuracyMxLr[idx])
        print("idx: ", idx, " Accuracy Score MNB: ", accuracyMxMnb[idx])


        # Increase the counter
        idx = idx + 1

    # Print average accuracy score for each algorithm
    print("\n")
    print("Average Accuracy Score SVC: ", accuracyMxSvc.mean())
    print("Average Accuracy Score RFC: ", accuracyMxRfc.mean())
    print("Average Accuracy Score LR: ",  accuracyMxLr.mean())
    print("Average Accuracy Score MNB: ", accuracyMxMnb.mean())

if __name__ == "__main__":
    main()
```

Running the Code

- The code can be simply run in Jupyter notebook by selecting “Cell” and then “Run All”.

- Input filename and number of comment to be considered are fixed and set inside the “Main”.



Word Cloud

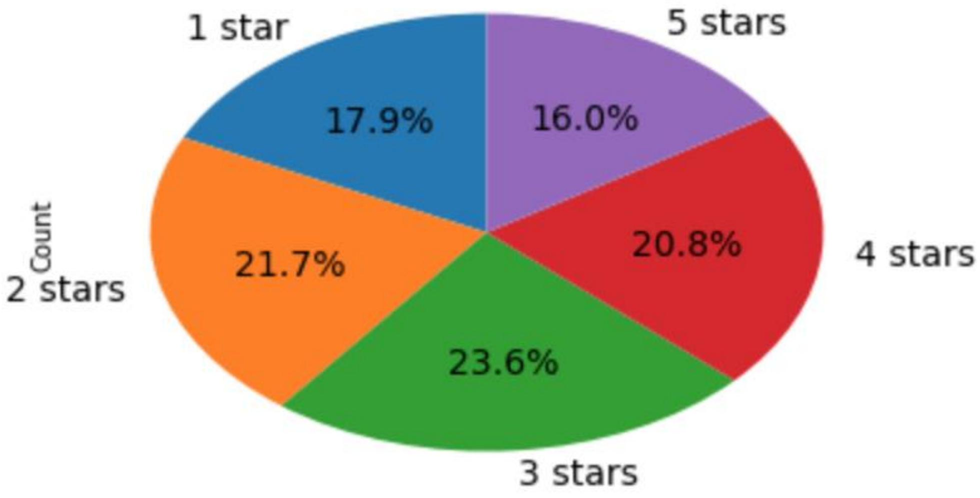
- Word cloud for 1-star reviews

- Word cloud for 5-star reviews

- Notice that likelihood of observin word "good" is high in 5-star reviews.

Pie Chart

- 5-star reviews are least frequent reviews (16%)
- 3-star reviews are most frequent reviews (23.6%)
- All other classes are somewhere in between



Balancing Classes

- After balancing classes, we consider 1601 reviews in each classs

```
('Number of comments in each class before balancing: ', Counter({'3': 2361, '2': 2168, '4': 2082, '1': 1788, '5': 160
1}))
('Number of comments in each class after balancing: ', Counter({'1': 1601, '3': 1601, '2': 1601, '5': 1601, '4': 1601
}))
```

Accuracy Scores

- 10,000 comments are considered.
- 10-fold cross-validation is used.
- SVC and LR have the best accuracy (~0.53)
- Although 53% may look low, it is much better than random classification which gives accuracy of 1/5 = 20% on average

```
...
('idx: ', 0, ' Accuracy Score SVC: ', 0.52933832709113604)
('idx: ', 0, ' Accuracy Score RFC: ', 0.41697877652933835)
('idx: ', 0, ' Accuracy Score LR: ', 0.51935081148564299)
('idx: ', 0, ' Accuracy Score MNB: ', 0.47565543071161048)
('idx: ', 1, ' Accuracy Score SVC: ', 0.50312109862671661)
('idx: ', 1, ' Accuracy Score RFC: ', 0.37952559300873906)
('idx: ', 1, ' Accuracy Score LR: ', 0.51061173533083648)
('idx: ', 1, ' Accuracy Score MNB: ', 0.45068664169787764)
('idx: ', 2, ' Accuracy Score SVC: ', 0.51935081148564299)
('idx: ', 2, ' Accuracy Score RFC: ', 0.36953807740324596)
('idx: ', 2, ' Accuracy Score LR: ', 0.52309612983770282)
('idx: ', 2, ' Accuracy Score MNB: ', 0.5168539325842697)
('idx: ', 3, ' Accuracy Score SVC: ', 0.54307116104868913)
('idx: ', 3, ' Accuracy Score RFC: ', 0.40699126092384519)
('idx: ', 3, ' Accuracy Score LR: ', 0.54556803995006242)
('idx: ', 3, ' Accuracy Score MNB: ', 0.50312109862671661)
('idx: ', 4, ' Accuracy Score SVC: ', 0.52684144818976275)
('idx: ', 4, ' Accuracy Score RFC: ', 0.37702871410736577)
('idx: ', 4, ' Accuracy Score LR: ', 0.52684144818976275)
('idx: ', 4, ' Accuracy Score MNB: ', 0.4706616729088639)
('idx: ', 5, ' Accuracy Score SVC: ', 0.51000000000000001)
('idx: ', 5, ' Accuracy Score RFC: ', 0.34875)
('idx: ', 5, ' Accuracy Score LR: ', 0.50875000000000004)
('idx: ', 5, ' Accuracy Score MNB: ', 0.51124999999999998)
('idx: ', 6, ' Accuracy Score SVC: ', 0.50875000000000004)
('idx: ', 6, ' Accuracy Score RFC: ', 0.42875000000000002)
('idx: ', 6, ' Accuracy Score LR: ', 0.52124999999999999)
('idx: ', 6, ' Accuracy Score MNB: ', 0.50875000000000004)
('idx: ', 7, ' Accuracy Score SVC: ', 0.53125)
('idx: ', 7, ' Accuracy Score RFC: ', 0.42875000000000002)
('idx: ', 7, ' Accuracy Score LR: ', 0.52749999999999997)
('idx: ', 7, ' Accuracy Score MNB: ', 0.52500000000000002)
('idx: ', 8, ' Accuracy Score SVC: ', 0.49375000000000002)
('idx: ', 8, ' Accuracy Score RFC: ', 0.40250000000000002)
('idx: ', 8, ' Accuracy Score LR: ', 0.49125000000000002)
('idx: ', 8, ' Accuracy Score MNB: ', 0.49125000000000002)
('idx: ', 9, ' Accuracy Score SVC: ', 0.61875000000000002)
('idx: ', 9, ' Accuracy Score RFC: ', 0.45250000000000001)
('idx: ', 9, ' Accuracy Score LR: ', 0.58999999999999997)
('idx: ', 9, ' Accuracy Score MNB: ', 0.35875000000000001)


('Average Accuracy Score SVC: ', 0.52842228464419483)
('Average Accuracy Score RFC: ', 0.40113124219725338)
('Average Accuracy Score LR: ', 0.52642181647940078)
('Average Accuracy Score MNB: ', 0.48119787765293387)
```

Optimization

- 100,000 comments are considered.

- Example 1: Linear Support Vector

Optimal value

Penalty parameter of error term (C)	0.01	0.1	0.2	0.4	0.6	0.8	1
Average Accuracy	0.55	0.594	0.595	0.592	0.587	0.584	0.582

- Example 2: Multinomial Naive Bayes

Optimal value

Additive smoothing parameter (alpha)	0.01	0.1	0.2	0.4	0.6	0.8	1
Average Accuracy	0.53	0.54	0.53	0.52	0.51	0.51	0.51