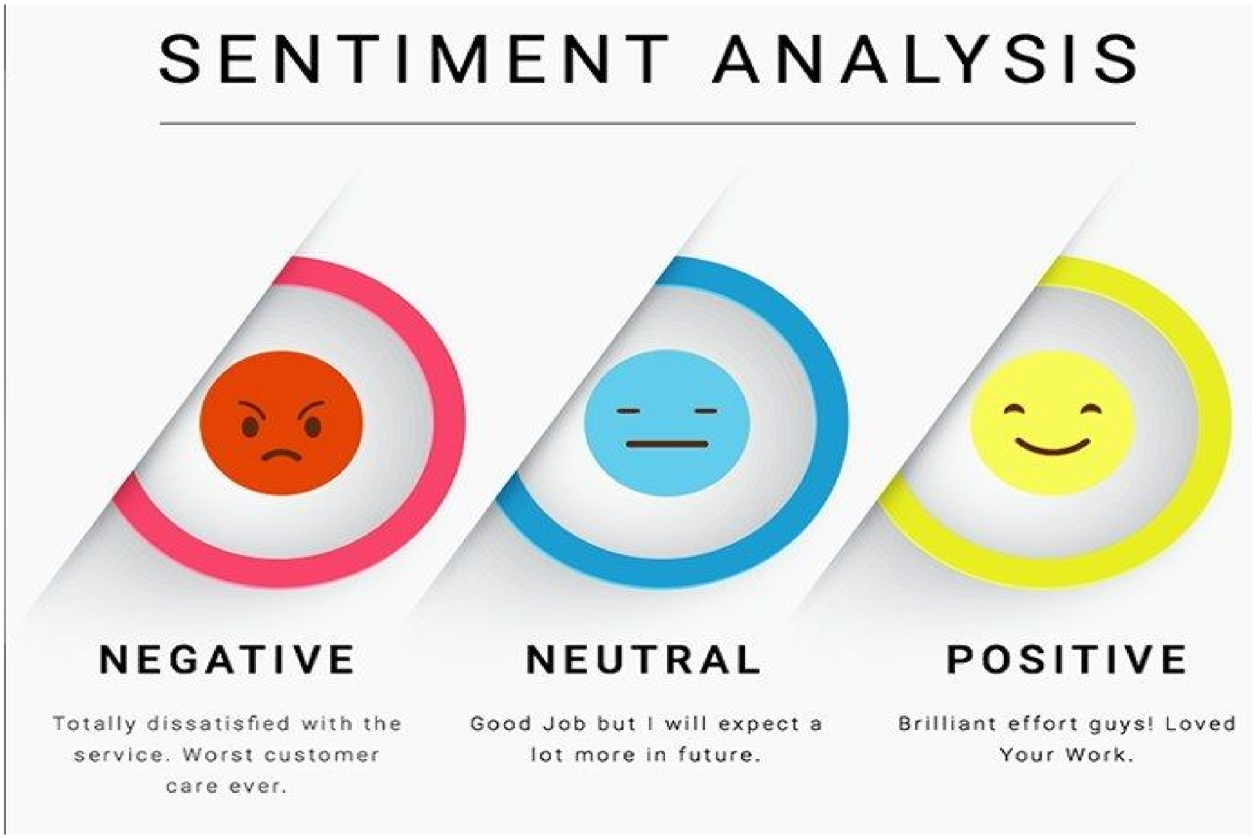

SENTIMENT ANALYSIS
NEGATIVE
Totally dissatisfied with the service. Worst customer care ever.
NEUTRAL
Good Job but I will expect a lot more in future.
POSITIVE
Brilliant effort guys! Loved Your Work.

Sentiment Analysis to figure out Deceptive Reviews on Hotel Dataset

Submitted By - Group 6

Aastha Kumar
Varun Jindal
Thejaswi Kampalli

Submitted To

Professor Sanket Tavarageri
San Jose State University

Overview

Recent years have brought the blasted ubiquity of community portals. Alongside of the content created by the portal editors users provide their input through discussions and personal notes in various social web spaces on mass scale, leave comments and reviews of products. Sentiment analysis is an attempt to take an advantage of vast amount of user generated content. It employs computing power to formalize the knowledge taken from user opinion and analyze it for further reuse. Our work deals with classifying the user sentiments on hotel reviews as negative or positive by eliminating deceptive reviews using supervised machine learning.

Dataset Description

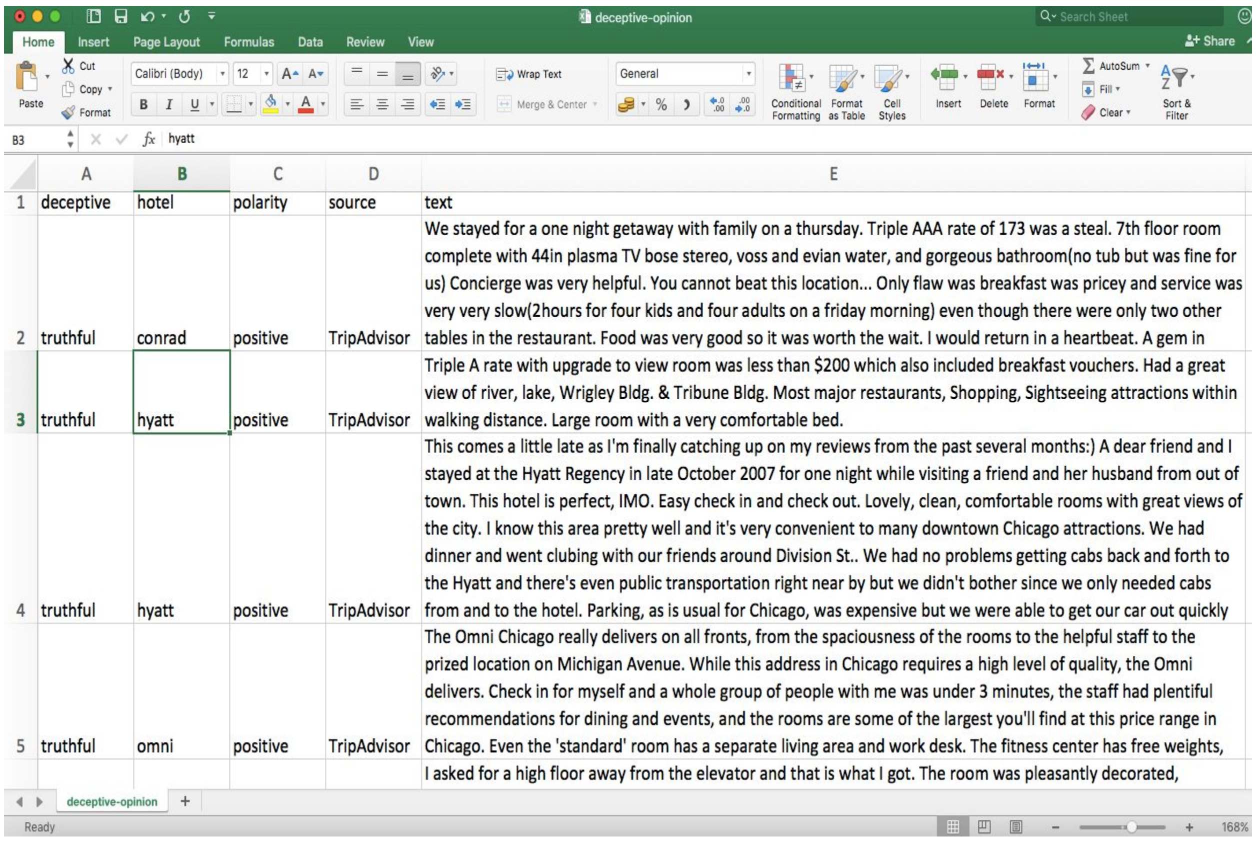
We picked up deceptive opinion dataset of a hotel from Kaggle.
https://www.kaggle.com/rtatman/deceptive-opinion-spam-corpus

Data is present in csv format consisting of 1600 records. The dataset contains reviews for 20 hotels from Chicago. This dataset contains:

➔ 400 truthful positive reviews from TripAdvisor.
➔ 400 deceptive positive reviews from Mechanical Turk.
➔ 400 truthful negative reviews from Expedia, Hotels.com, Orbitz, Priceline, TripAdvisor and Yelp.
➔ 400 deceptive negative reviews from Mechanical Turk.

Note:

➔ Deceptive review means a fake reviews where the authenticity of the review is not verified and truthful review indicates it is a genuine review.
➔ Positive review means that the user is appreciating the hotel and negative means user is not happy with hotel or its services.

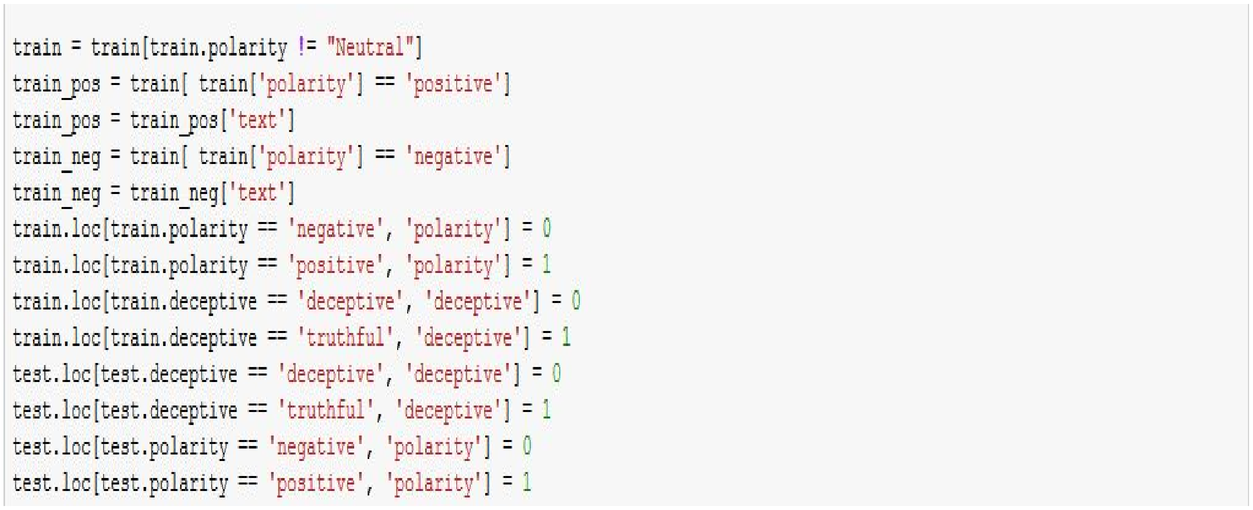


	A	B	C	D	E
1	deceptive	hotel	polarity	source	text
2	truthful	conrad	positive	TripAdvisor	We stayed for a one night getaway with family on a thursday. Triple AAA rate of 173 was a steal. 7th floor room complete with 44in plasma TV bose stereo, voss and evian water, and gorgeous bathroom(no tub but was fine for us) Concierge was very helpful. You cannot beat this location... Only flaw was breakfast was pricey and service was very very slow(2hours for four kids and four adults on a friday morning) even though there were only two other tables in the restaurant. Food was very good so it was worth the wait. I would return in a heartbeat. A gem in
3	truthful	hyatt	positive	TripAdvisor	Triple A rate with upgrade to view room was less than $200 which also included breakfast vouchers. Had a great view of river, lake, Wrigley Bldg. & Tribune Bldg. Most major restaurants, Shopping, Sightseeing attractions within walking distance. Large room with a very comfortable bed.
4	truthful	hyatt	positive	TripAdvisor	This comes a little late as I'm finally catching up on my reviews from the past several months:) A dear friend and I stayed at the Hyatt Regency in late October 2007 for one night while visiting a friend and her husband from out of town. This hotel is perfect, IMO. Easy check in and check out. Lovely, clean, comfortable rooms with great views of the city. I know this area pretty well and it's very convenient to many downtown Chicago attractions. We had dinner and went clubing with our friends around Division St.. We had no problems getting cabs back and forth to the Hyatt and there's even public transportation right near by but we didn't bother since we only needed cabs from and to the hotel. Parking, as is usual for Chicago, was expensive but we were able to get our car out quickly
5	truthful	omni	positive	TripAdvisor	The Omni Chicago really delivers on all fronts, from the spaciousness of the rooms to the helpful staff to the prized location on Michigan Avenue. While this address in Chicago requires a high level of quality, the Omni delivers. Check in for myself and a whole group of people with me was under 3 minutes, the staff had plentiful recommendations for dining and events, and the rooms are some of the largest you'll find at this price range in Chicago. Even the 'standard' room has a separate living area and work desk. The fitness center has free weights,
					I asked for a high floor away from the elevator and that is what I got. The room was pleasantly decorated,

Each of the above dataset consists of 20 reviews for each of the 20 most popular Chicago hotels.
The data set has the following fields:-

- ★ Deceptive - Contains values as “truthful” and “deceptive”. This column help us in classifying the reviews as genuine or fake.
- ★ Hotel - name of the hotel.
- ★ Polarity - Contains values as positive or negative. This column help us in determining the sentiments of users.
- ★ Source - where the review was taken from
- ★ Text - the actual review

Data Cleaning and Feature Engineering



```
train = train[train.polarity != "Neutral"]
train_pos = train[ train['polarity'] == 'positive']
train_pos = train_pos['text']
train_neg = train[ train['polarity'] == 'negative']
train_neg = train_neg['text']
train.loc[train.polarity == 'negative', 'polarity'] = 0
train.loc[train.polarity == 'positive', 'polarity'] = 1
train.loc[train.deceptive == 'deceptive', 'deceptive'] = 0
train.loc[train.deceptive == 'truthful', 'deceptive'] = 1
test.loc[test.deceptive == 'deceptive', 'deceptive'] = 0
test.loc[test.deceptive == 'truthful', 'deceptive'] = 1
test.loc[test.polarity == 'negative', 'polarity'] = 0
test.loc[test.polarity == 'positive', 'polarity'] = 1
```

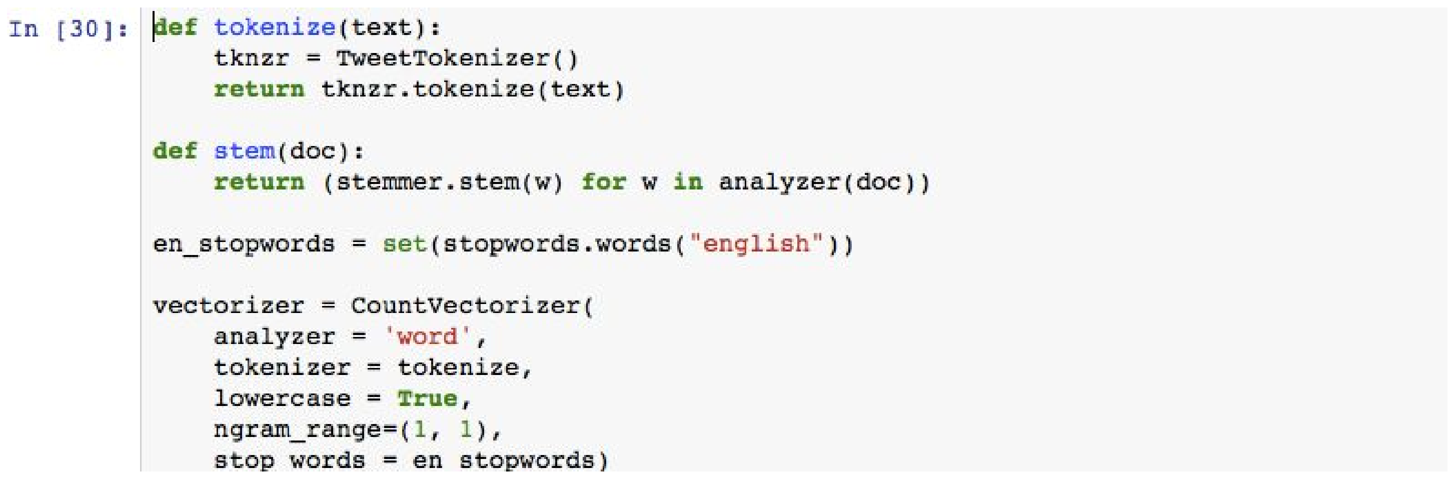
We collected the data from deceptive-opinion.csv into pandas dataframe and selected **Polarity**, **Deceptive** and **Text** columns as the features to work on.

Data Cleaning -

Tokenization :- Tokenization is the process of chopping sentences in array or words or tokens. We defined a tokenize function to break the review sentences into tokens. We have used TweetTokenizer function present in nltk.tokenize package to tokenize the review text.

TweetTokenizer performs some preprocessing by removing entities from text by converting them to their corresponding unicode character through the _replace_html_entities() function. It also removes username handles using the remove_handles() function (if exists). Optionally, it normalize the word length through the reduce_lengthening function. Then, shortens the *problematic sequences of characters* using the HANG_RE regex. Lastly, the actual tokenization takes place through the WORD_RE regex. After the WORD_RE regex, it optionally preserve the case of emoticons before lowercasing the tokenized output.

Word Normalization :- is the reduction of each word to its base/stem form (by chopping of the affixes). While performing normalization few things that should be considered are like Capital letters should be normalized to lowercase, stemming etc. stem function performs that task.

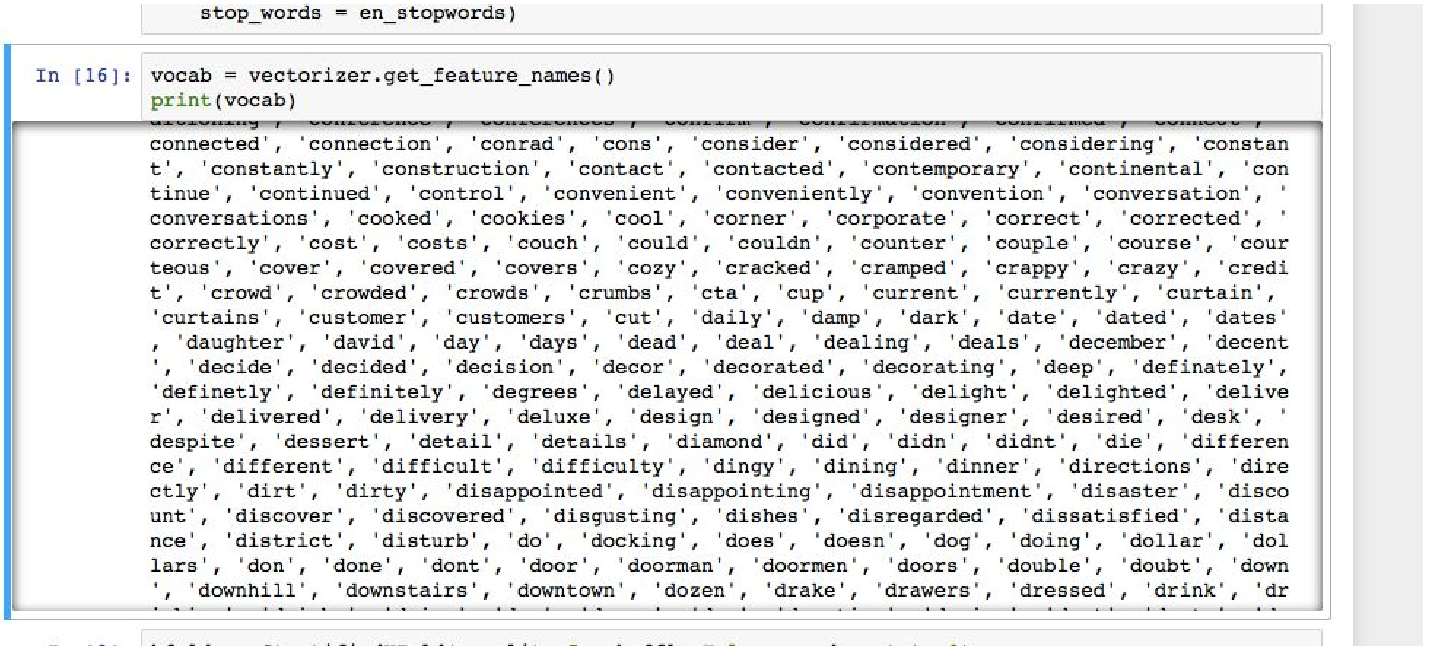


```
In [30]: def tokenize(text):
             tknzr = TweetTokenizer()
             return tknzr.tokenize(text)

         def stem(doc):
             return (stemmer.stem(w) for w in analyzer(doc))

         en_stopwords = set(stopwords.words("english"))

         vectorizer = CountVectorizer(
             analyzer = 'word',
             tokenizer = tokenize,
             lowercase = True,
             ngram_range=(1, 1),
             stop_words = en_stopwords)
```

Feature Engineering :-

Bag of Words :-We used feature_extraction module from scikit-learn to create bag-of-words features (as shown in above screenshot)

A feature or aspect is an attribute or component of an entity, for example a hotel can have a convenient location, but mediocre food. This problem involves several sub-problems, e.g., identifying relevant entities, extracting their features/aspects, and determining whether an opinion expressed on each feature/aspect is positive, negative or neutral. So all the words in text column were the features in determining the sentiments of the user. For example some of the important features in determining the polarity of the review were as following like dirty, rude, disappointed is more likely to be negative while review containing words like enjoyed, fabulous, delicious is more likely to be positive.



```
    stop_words = en_stopwords)

In [16]: vocab = vectorizer.get_feature_names()
         print(vocab)

connected', 'connection', 'conrad', 'cons', 'consider', 'considered', 'considering', 'constan
t', 'constantly', 'construction', 'contact', 'contacted', 'contemporary', 'continental', 'con
tinue', 'continued', 'control', 'convenient', 'conveniently', 'convention', 'conversation', '
conversations', 'cooked', 'cookies', 'cool', 'corner', 'corporate', 'correct', 'corrected', '
correctly', 'cost', 'costs', 'couch', 'could', 'couldn', 'counter', 'couple', 'course', 'cour
teous', 'cover', 'covered', 'covers', 'cozy', 'cracked', 'cramped', 'crappy', 'crazy', 'credi
t', 'crowd', 'crowded', 'crowds', 'crumbs', 'cta', 'cup', 'current', 'currently', 'curtain',
'curtains', 'customer', 'customers', 'cut', 'daily', 'damp', 'dark', 'date', 'dated', 'dates'
, 'daughter', 'david', 'day', 'days', 'dead', 'deal', 'dealing', 'deals', 'december', 'decent
', 'decide', 'decided', 'decision', 'decor', 'decorated', 'decorating', 'deep', 'definately',
'definetly', 'definitely', 'degrees', 'delayed', 'delicious', 'delight', 'delighted', 'delive
r', 'delivered', 'delivery', 'deluxe', 'design', 'designed', 'designer', 'desired', 'desk', '
despite', 'dessert', 'detail', 'details', 'diamond', 'did', 'didn', 'didnt', 'die', 'differen
ce', 'different', 'difficult', 'difficulty', 'dingy', 'dining', 'dinner', 'directions', 'dire
ctly', 'dirt', 'dirty', 'disappointed', 'disappointing', 'disappointment', 'disaster', 'disco
unt', 'discover', 'discovered', 'disgusting', 'dishes', 'disregarded', 'dissatisfied', 'dista
nce', 'district', 'disturb', 'do', 'docking', 'does', 'doesn', 'dog', 'doing', 'dollar', 'dol
lars', 'don', 'done', 'dont', 'door', 'doorman', 'doormen', 'doors', 'double', 'doubt', 'down
', 'downhill', 'downstairs', 'downtown', 'dozen', 'drake', 'drawers', 'dressed', 'drink', 'dr
```

We defined stemming and tokenizing the text and used these definitions as a callable method in CountVectorizer and calculated the count of each word and formed a feature vector.

We performed filtering of truthful review and in order to do that we trained a model to predict a review's deceptiveness and used it to predict the deceptiveness of the review. We then eliminated all the reviews predicted as deceptive.

Classifier Evaluation :-

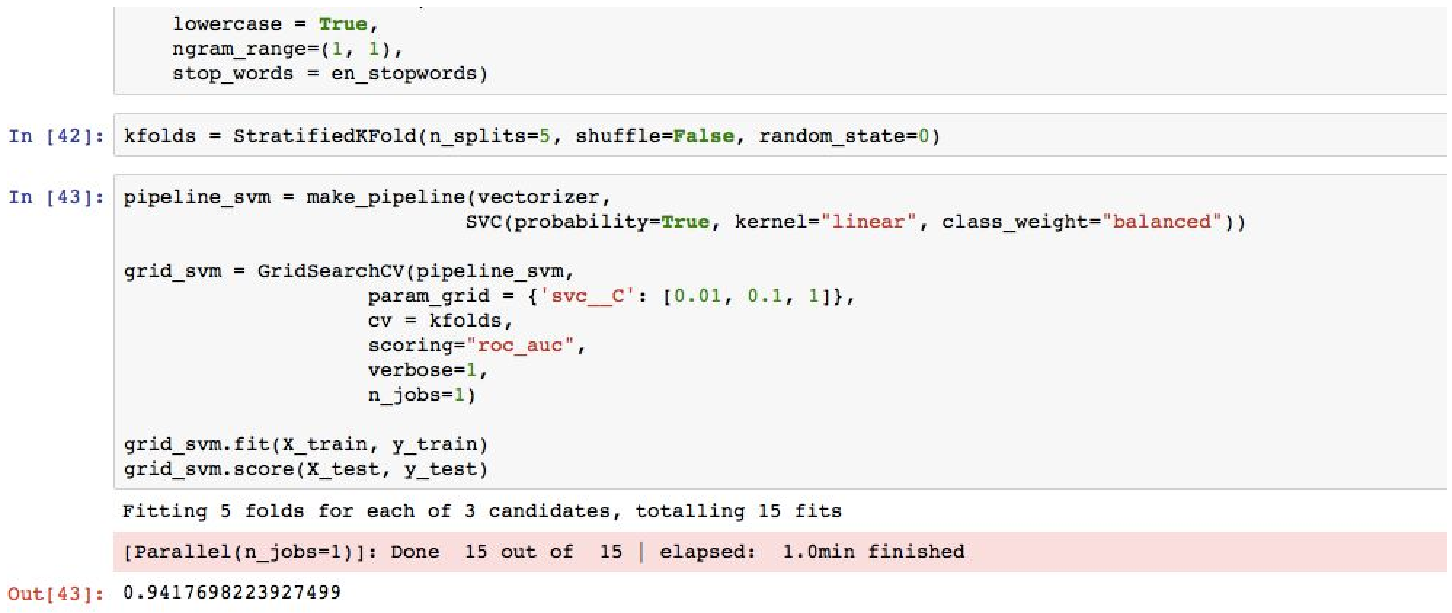
For determining the accuracy of a single Classifier, or comparing the results of different Classifier, the F-score is usually used. This F-score is given by

F = 2* (p * r)/ (p + r)

We are using F1 - score to compare different models.

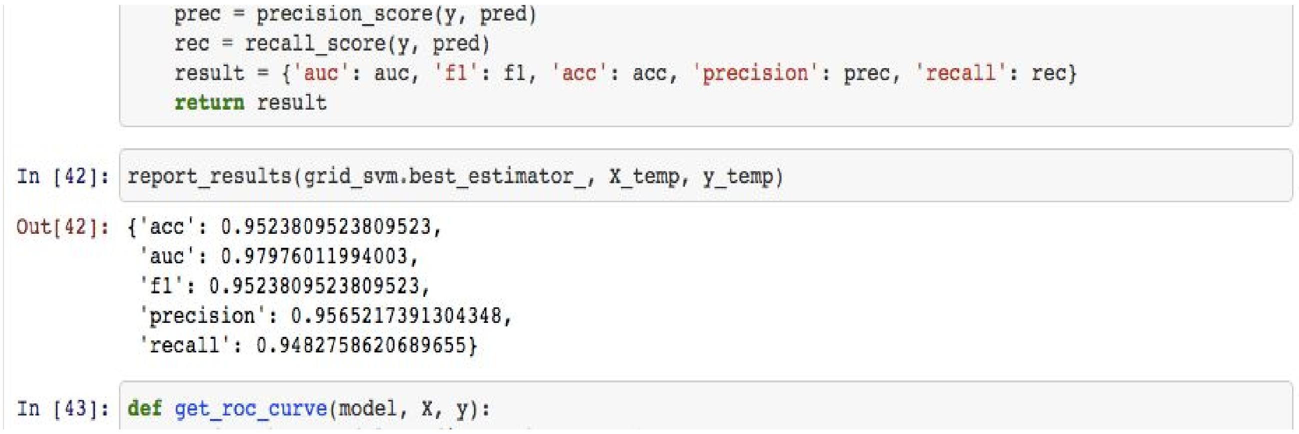
Training the Model :-

We used cross validation and grid search to find good hyperparameters for our SVM model. We built a pipeline to ignore features from the validation folds when building each training model.



```
            lowercase = True,
            ngram_range=(1, 1),
            stop_words = en_stopwords)

In [42]: kfolds = StratifiedKFold(n_splits=5, shuffle=False, random_state=0)

In [43]: pipeline_svm = make_pipeline(vectorizer,
                                     SVC(probability=True, kernel="linear", class_weight="balanced"))

         grid_svm = GridSearchCV(pipeline_svm,
                             param_grid = {'svc__C': [0.01, 0.1, 1]},
                             cv = kfolds,
                             scoring="roc_auc",
                             verbose=1,
                             n_jobs=1)

         grid_svm.fit(X_train, y_train)
         grid_svm.score(X_test, y_test)

         Fitting 5 folds for each of 3 candidates, totalling 15 fits

         [Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:  1.0min finished

Out[43]: 0.9417698223927499
```

Result of model (with the best hyperparameters) on the test data

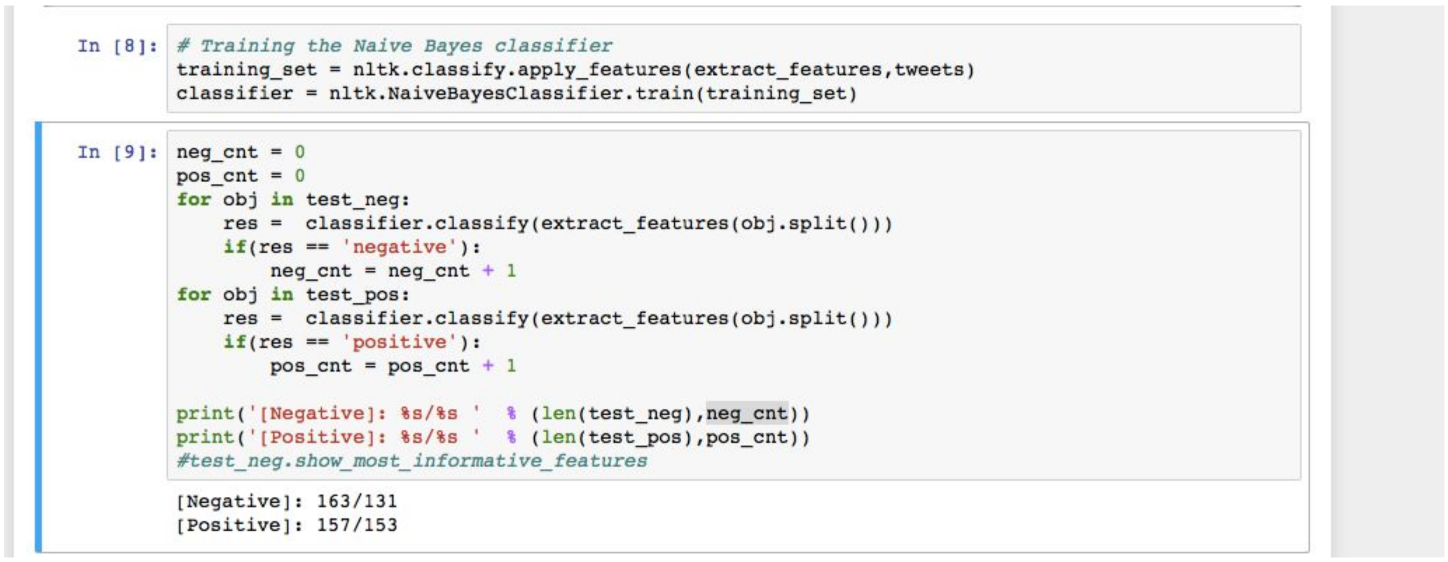


```
              prec = precision_score(y, pred)
              rec = recall_score(y, pred)
              result = {'auc': auc, 'f1': f1, 'acc': acc, 'precision': prec, 'recall': rec}
              return result

In [42]: report_results(grid_svm.best_estimator_, X_temp, y_temp)

Out[42]: {'acc': 0.9523809523809523,
          'auc': 0.97976011994003,
          'f1': 0.9523809523809523,
          'precision': 0.9565217391304348,
          'recall': 0.9482758620689655}

In [43]: def get_roc_curve(model, X, y):
```

Machine Learning Model selected

Naive Bayes Approach

We started our project by first training our model using Naive Bayes approach. However the model didn't perform that well. Only 88% of reviews were classified correctly.

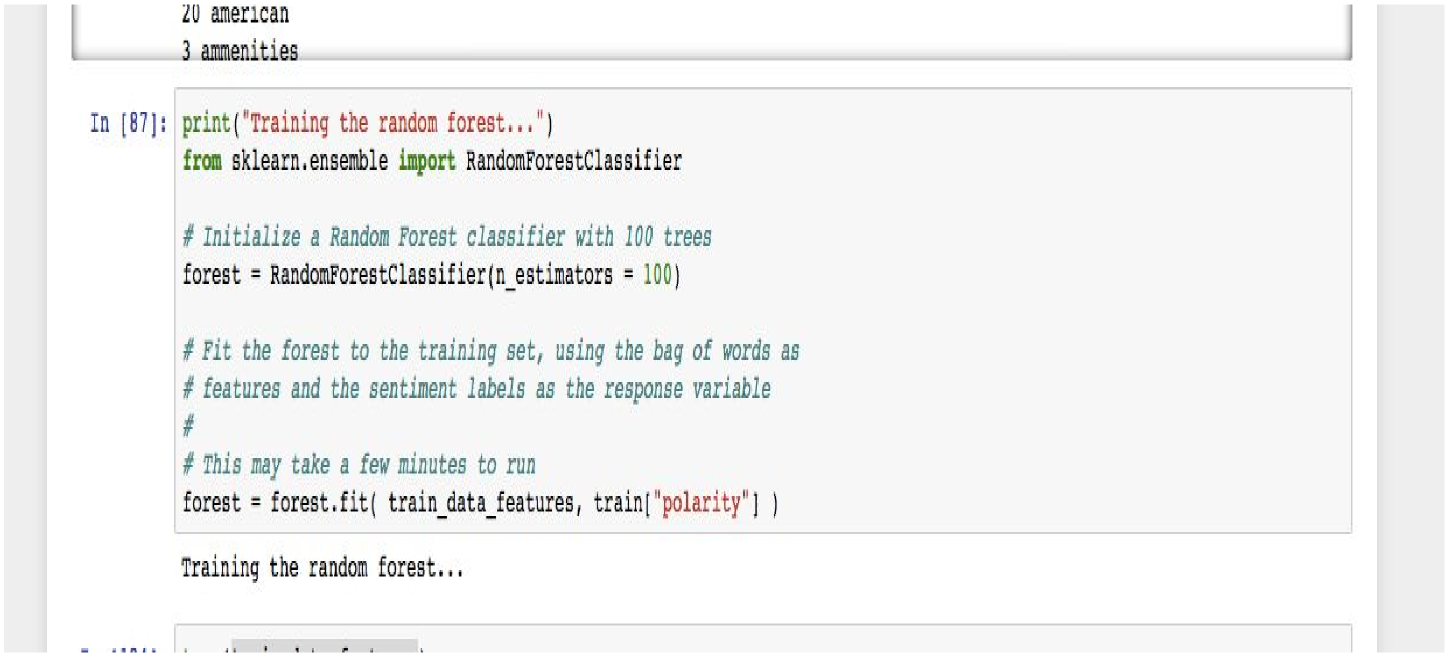


```
In [8]: # Training the Naive Bayes classifier
        training_set = nltk.classify.apply_features(extract_features,tweets)
        classifier = nltk.NaiveBayesClassifier.train(training_set)
```

```
In [9]: neg_cnt = 0
        pos_cnt = 0
        for obj in test_neg:
            res =  classifier.classify(extract_features(obj.split()))
            if(res == 'negative'):
                neg_cnt = neg_cnt + 1
        for obj in test_pos:
            res =  classifier.classify(extract_features(obj.split()))
            if(res == 'positive'):
                pos_cnt = pos_cnt + 1

        print('[Negative]: %s/%s '  % (len(test_neg),neg_cnt))
        print('[Positive]: %s/%s '  % (len(test_pos),pos_cnt))
        #test_neg.show_most_informative_features

        [Negative]: 163/131
        [Positive]: 157/153
```

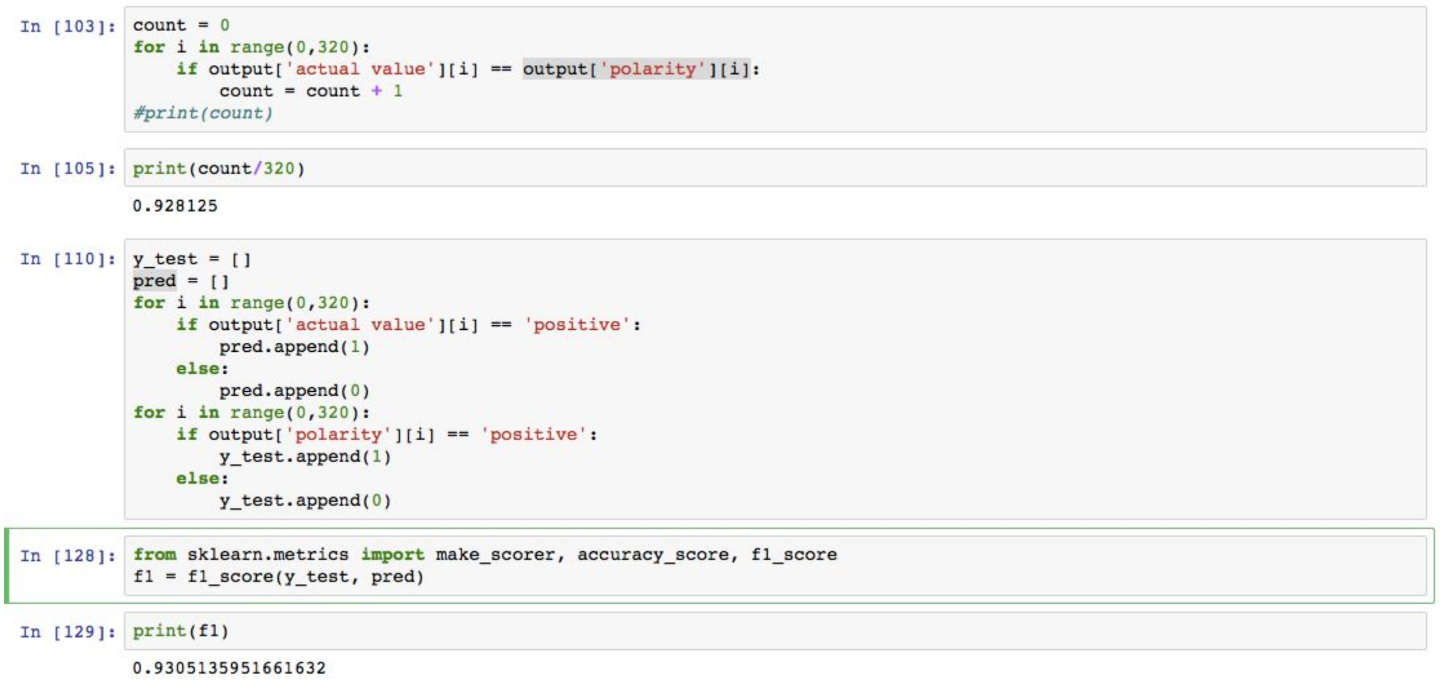
Random Forest Approach

Next we tried Random Forest, starting with cleaning the data i.e. removing stop words, numbers etc and creating bag of words. Next we did set the reasonable number of trees - 100 as the default value and tested the model. F1 score came out to be better than Naive Bayes approach but we continued our research to figure out better model and tried training the model with svm. Below screenshot just shows the F1 score achieved from random forest approach. For more details on implementation please check out the code for this implementation.
https://github.com/aasthakumar/CMPE-256-Project



```
20 american
3 ammenities
```

```
In [87]: print("Training the random forest...")
         from sklearn.ensemble import RandomForestClassifier

         # Initialize a Random Forest classifier with 100 trees
         forest = RandomForestClassifier(n_estimators = 100)

         # Fit the forest to the training set, using the bag of words as
         # features and the sentiment labels as the response variable
         #
         # This may take a few minutes to run
         forest = forest.fit( train_data_features, train["polarity"] )

         Training the random forest...
```

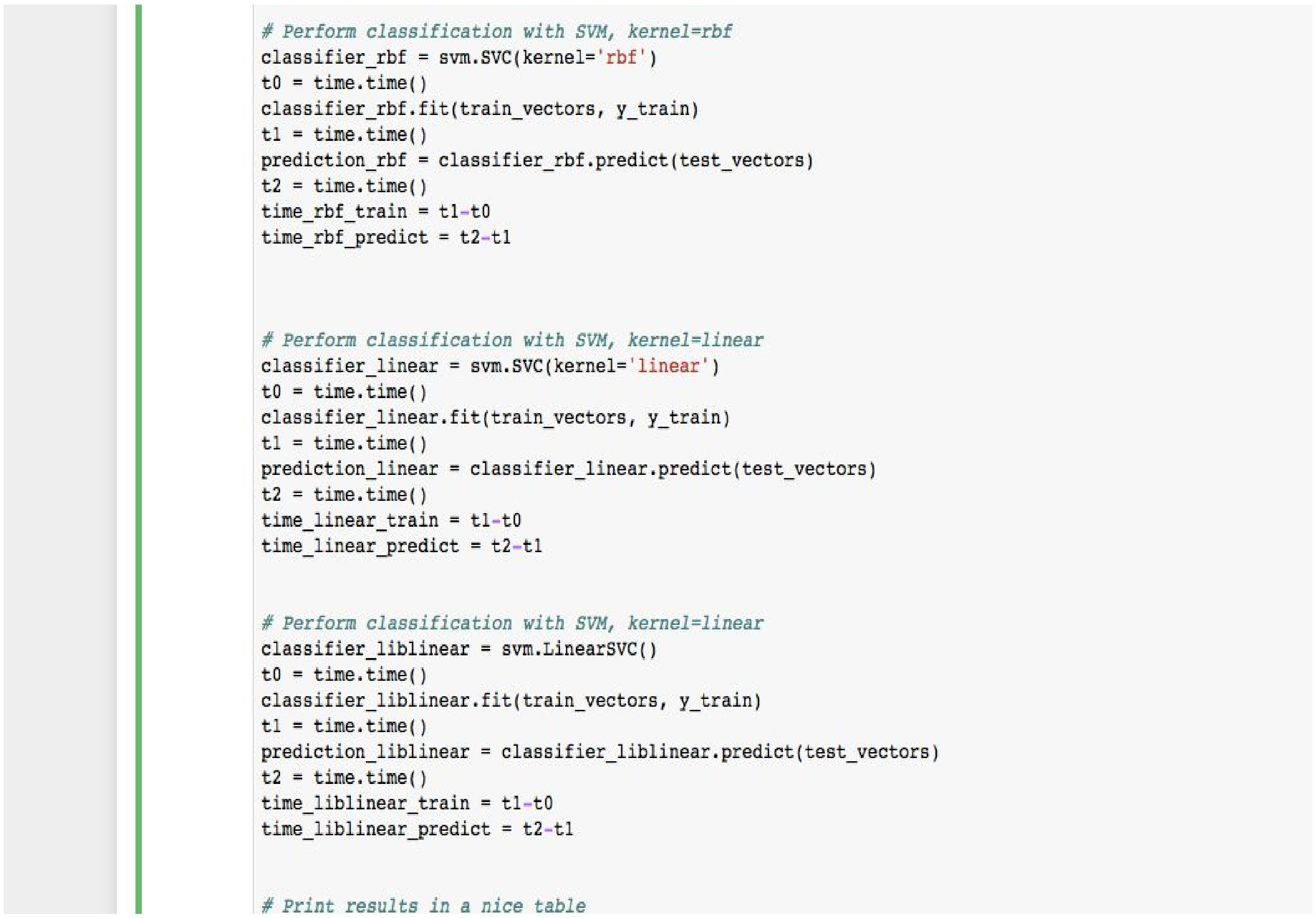


```
In [103]: count = 0
          for i in range(0,320):
              if output['actual value'][i] == output['polarity'][i]:
                  count = count + 1
          #print(count)

In [105]: print(count/320)

          0.928125

In [110]: y_test = []
          pred = []
          for i in range(0,320):
              if output['actual value'][i] == 'positive':
                  pred.append(1)
              else:
                  pred.append(0)
          for i in range(0,320):
              if output['polarity'][i] == 'positive':
                  y_test.append(1)
              else:
                  y_test.append(0)

In [128]: from sklearn.metrics import make_scorer, accuracy_score, f1_score
          f1 = f1_score(y_test, pred)

In [129]: print(f1)

          0.9305135951661632
```

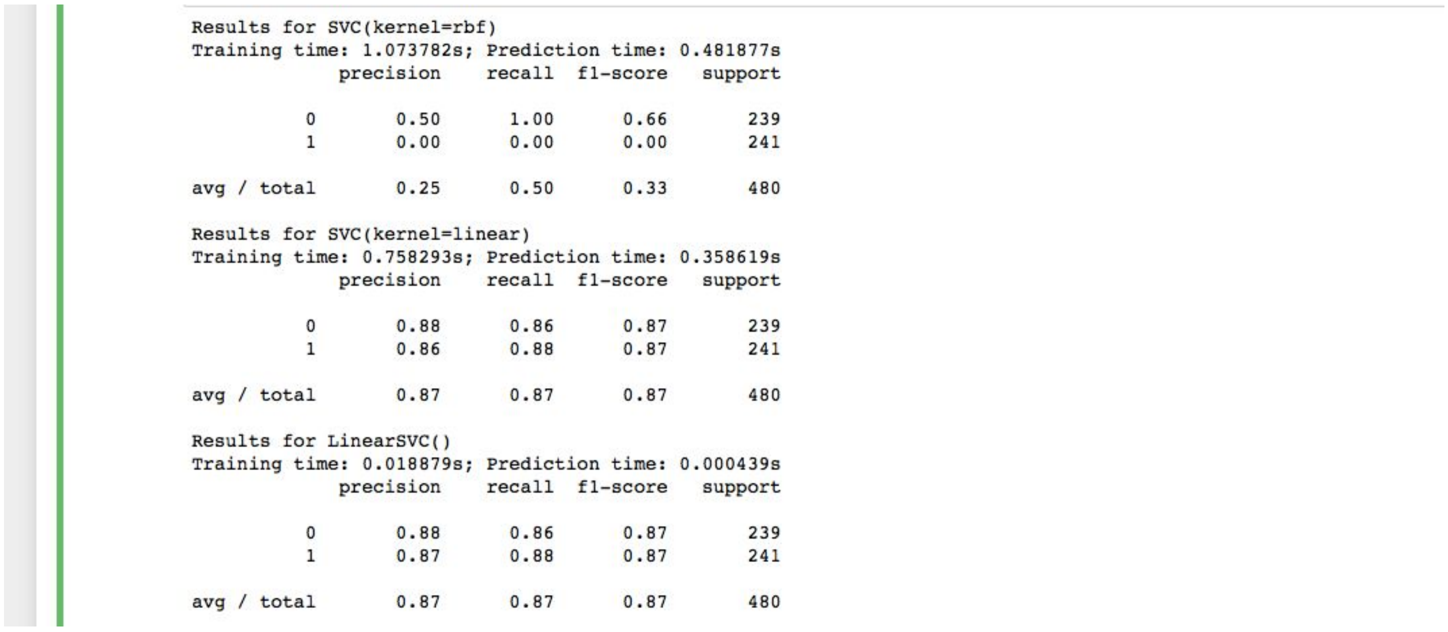
Support Vector Machine

Next we tried training our model with Support Vector Machine (SVM) which provides us with the better results. To improve on the results we further used Grid search CV which does an Exhaustive search over specified parameter values for an estimator. Our final f1 score came out to be 0.952 which is a really a good score for any text processing machine learning model.

We also tried training the model with different kernel tricks and compared the result. Training with rbf didn't produce good results while training with linear kernel trick turned out to be good.

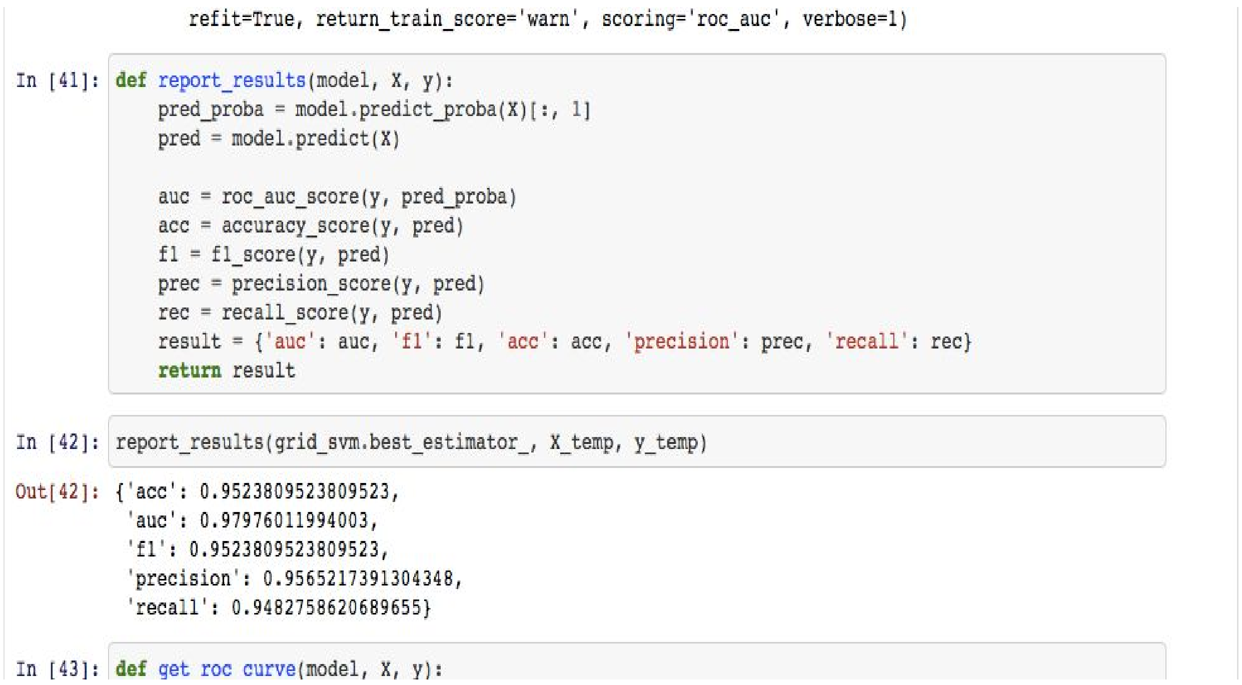


```python
# Perform classification with SVM, kernel=rbf
classifier_rbf = svm.SVC(kernel='rbf')
t0 = time.time()
classifier_rbf.fit(train_vectors, y_train)
t1 = time.time()
prediction_rbf = classifier_rbf.predict(test_vectors)
t2 = time.time()
time_rbf_train = t1-t0
time_rbf_predict = t2-t1



# Perform classification with SVM, kernel=linear
classifier_linear = svm.SVC(kernel='linear')
t0 = time.time()
classifier_linear.fit(train_vectors, y_train)
t1 = time.time()
prediction_linear = classifier_linear.predict(test_vectors)
t2 = time.time()
time_linear_train = t1-t0
time_linear_predict = t2-t1


# Perform classification with SVM, kernel=linear
classifier_liblinear = svm.LinearSVC()
t0 = time.time()
classifier_liblinear.fit(train_vectors, y_train)
t1 = time.time()
prediction_liblinear = classifier_liblinear.predict(test_vectors)
t2 = time.time()
time_liblinear_train = t1-t0
time_liblinear_predict = t2-t1


# Print results in a nice table
```



```
Results for SVC(kernel=rbf)
Training time: 1.073782s; Prediction time: 0.481877s
             precision    recall  f1-score   support

          0       0.50      1.00      0.66       239
          1       0.00      0.00      0.00       241

avg / total       0.25      0.50      0.33       480

Results for SVC(kernel=linear)
Training time: 0.758293s; Prediction time: 0.358619s
             precision    recall  f1-score   support

          0       0.88      0.86      0.87       239
          1       0.86      0.88      0.87       241

avg / total       0.87      0.87      0.87       480

Results for LinearSVC()
Training time: 0.018879s; Prediction time: 0.000439s
             precision    recall  f1-score   support

          0       0.88      0.86      0.87       239
          1       0.87      0.88      0.87       241

avg / total       0.87      0.87      0.87       480
```

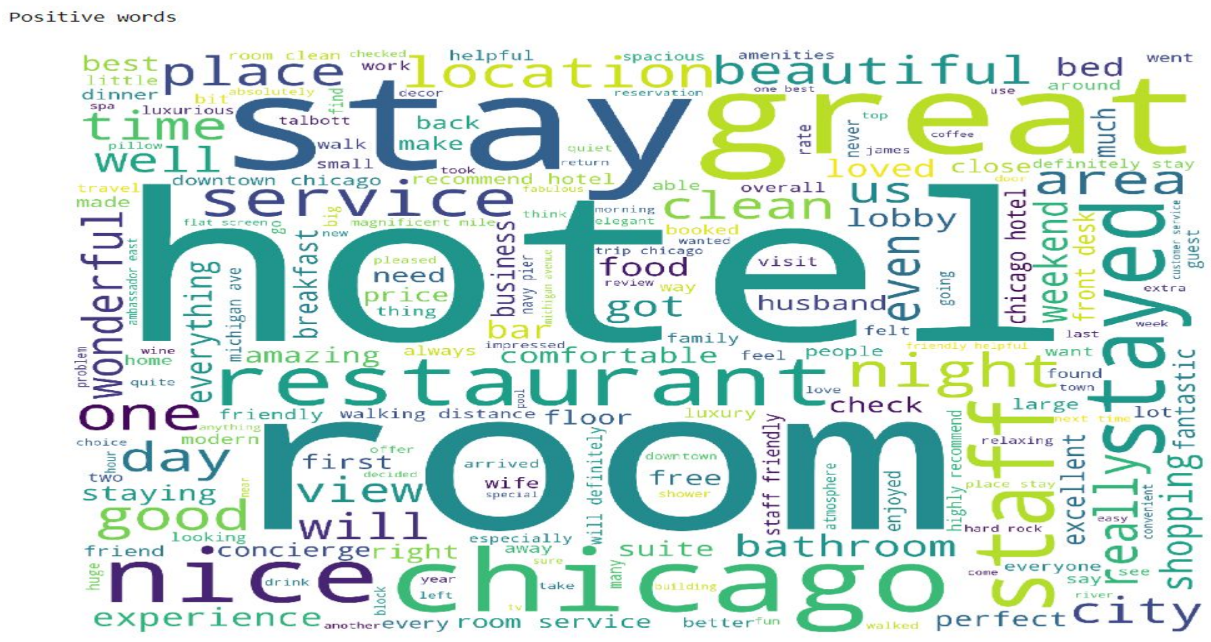
Result of SVM best hyperplane



```
            refit=True, return_train_score='warn', scoring='roc_auc', verbose=1)

In [41]: def report_results(model, X, y):
             pred_proba = model.predict_proba(X)[:, 1]
             pred = model.predict(X)

             auc = roc_auc_score(y, pred_proba)
             acc = accuracy_score(y, pred)
             f1 = f1_score(y, pred)
             prec = precision_score(y, pred)
             rec = recall_score(y, pred)
             result = {'auc': auc, 'f1': f1, 'acc': acc, 'precision': prec, 'recall': rec}
             return result

In [42]: report_results(grid_svm.best_estimator_, X_temp, y_temp)

Out[42]: {'acc': 0.9523809523809523,
          'auc': 0.97976011994003,
          'f1': 0.9523809523809523,
          'precision': 0.9565217391304348,
          'recall': 0.9482758620689655}

In [43]: def get_roc_curve(model, X, y):
```

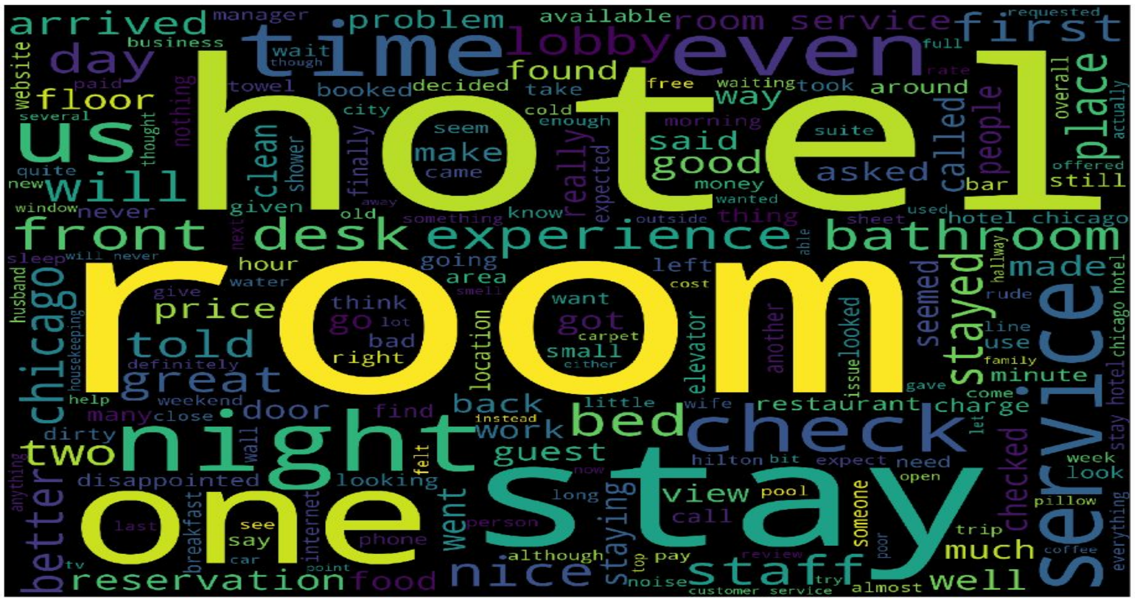
Results

Below you can see the list of positive words. Larger the word size higher the frequency of that word in the reviews given by user.

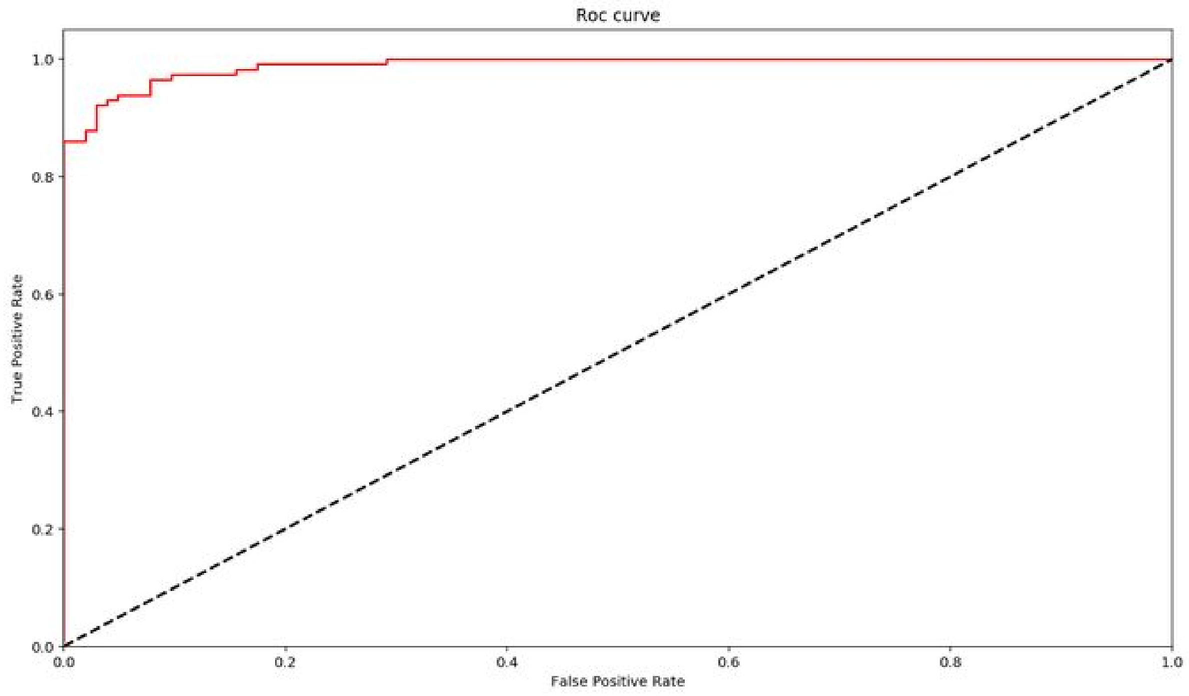


Below you can see the list of negative words. Larger the word size higher the frequency of that word in the reviews given by user.

```
Negative words
```



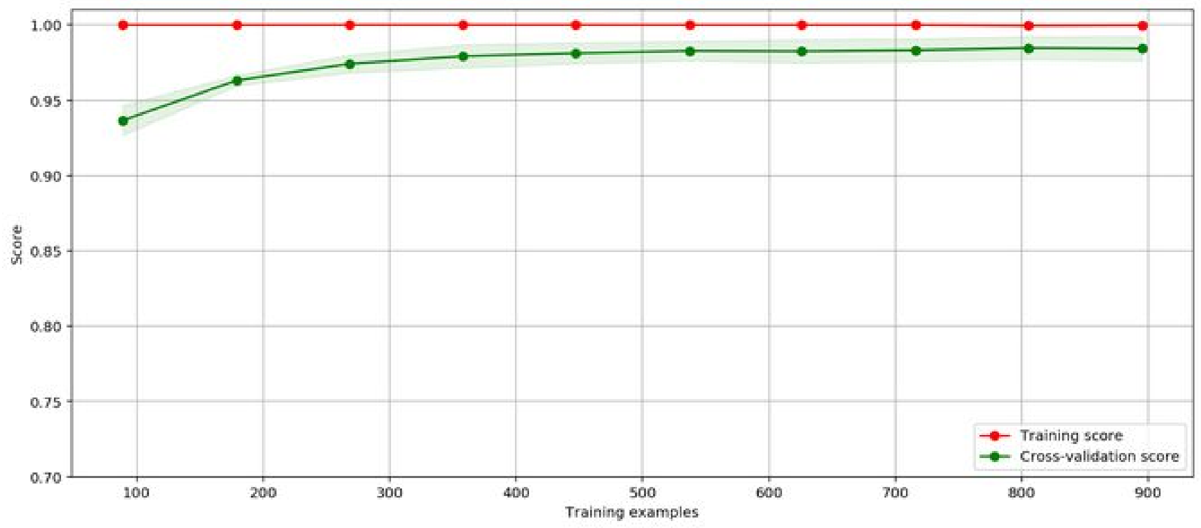
Below is the Receiver operator characteristic curve for the machine learning model we created.



Final accuracy of our model.

```
{'acc': 0.9523809523809523,
 'auc': 0.97976011994003,
 'f1': 0.9523809523809523,
 'precision': 0.9565217391304348,
 'recall': 0.9482758620689655}
```

Training score vs cross validation score.



Contributions of each member:

SNo.	Name	Contribution
1	Aastha Kumar	Model selection and model tuning for classifying deceptiveness of reviews. Worked on project report.
2	Varun Jindal	Model selection and model tuning for classifying polarity of reviews. Worked on project report.
3	Thejaswi kampalli	Data cleaning, data preprocessing, data visualization and Worked on project report.