Individual Final Report
Renzo P. Castagnino

1. Introduction

The project aims to predict the hotel rating based on the reviews. In order to achieve this, we used text mining techniques to extract information from the reviews and accurately predict the rating. We used a data set of 10,000 instances of US hotels reviews from 2016 to 2018.

To achieve the results, first we depurated the data by eliminating unnecessary columns or features, and also cleaning the data by removing null values, noise reduction, lower casing. Furthermore, In the preprocessing stage, we also performed some techniques specifically for text mining that includes remove stop words, count rare words, tokenization, lemmatization and stemming.

The models used to predict the ratings were Naïve Bayes, Support Vector Machine, and Linear Vector Machine with Lasso penalty. In all 3 cases, the models were able to predict 50% of the data. We could conclude that more data will be needed to increase the accuracy of the models.

2. Description of individual work

The model used was Linear Vector Machine with Lasso Regression. It is expected to have a better result by using LASSO to optimize the tuning parameter. With this model, we are trying to improve the results obtain by the support vector machine, and have a better accuracy in the predictive model. The basic (linear) maximum margin program is defined by the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b,\xi_i} \ & \frac{1}{2}\ \|\mathbf{w}\|_2^2 + C\sum_{i=1}^{m}\xi_i \\ s.t.\ & y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i - b) \geq 1 - \xi_i \qquad\qquad i = 1, ..., m \\ & \xi_i \geq 0 \end{aligned}$$

This program finds a hyperplane between the groups of data points and uses that to categorize new data. In this model, we are making use of two additional models: Chi-squared and lasso penalty for feature tuning. LASSO solves the following optimization problem where t is a tuning parameter:

$$\min \|X\beta - b\|_2^2 \qquad (1)$$

$$s.t.\ \sum_{j=1}^{d}|\beta_j| \leq t \qquad (2)$$

LASSO will set some coefficients βj = 0, Shrinks the parameters (and variance) and introduces bias to decrease mean squared error.

In the case of the Chi-squared, it was used to select the best features out of the bag of words to train our model. Using k=10,000 to select the best 10,000 features, and by giving weighting to each of the features.

In regards of the mode, our test size is 25% and the vectorization of n-grams was from 1-3. It is also valid to mention that for this model, since we are trying to predict the rating of the hotel based on the reviews, we had to standardize the rating values. In the data set, rating could be decimal numbers (such as 1.2, 1.5, 4.8, etc.). For this model, that feature was round to improve the accuracy of the model.

3. Description of work / code

```
# Renzo: delete NaN and reindex
self.pr_df['reviews_text'] = self.pr_df['reviews_text'].dropna().reset_index(drop=True)
```

```
# This code will remove the stop words from the reviews such as “the”, “a”, “an”, “in”)

def remove_stop_w(self):
    stop_words = stopwords.words('english')
    self.pr_df["reviews_text"] = self.pr_df["reviews_text"].apply(
        lambda x: " ".join(x for x in x.split() if x not in stop_words))
    # Cristina. Get ride of numbers
    self.pr_df["reviews_text"] = self.pr_df["reviews_text"].str.replace('\d+', '')
    self.pr_df['reviews_text'] = self.pr_df['reviews_text'].dropna().reset_index(
        drop=True)  # Renzo: delete NaN and reindex
    return self.pr_df
```

This code will allow to count all the rare words to select how many we will delete. We conclude to delete the 15 most common words (rather that 10 or 20)

```
def count_rare_word(self):
    freq = pd.Series(" ".join(self.pr_df['reviews_text']).split()).value_counts()[-15:]
    # freq = pd.Series(' '.join(train['tweet']).split()).value_counts()[-10:]
    return freq
```

Remove the rare words from the reviews column
```
def remove_rare_words(self):

    freq = pd.Series(" ".join(self.pr_df['reviews_text']).split()).value_counts()[-15:]
    freq = list(freq.index)
    self.pr_df["reviews_text"] = self.pr_df["reviews_text"].apply(
        lambda x: " ".join(x for x in x.split() if x not in freq))
    self.pr_df['reviews_text'] = self.pr_df['reviews_text'].dropna().reset_index(
        drop=True)  # Renzo: delete NaN and reindex

    return self.pr_df
```

Converts the word into its root word

```
    def lematization(self):
        self.pr_df["reviews_text_lematized"] = self.pr_df["reviews_text"]
        self.pr_df["reviews_text_lematized"] = self.pr_df["reviews_text_lematized"].apply(lambda
x: " ".join([Word(word).lemmatize() for word in x.split()]))
        return self.pr_df
```

Cutts off the end or the beginning of the word, taking into account a list of common prefixes and suffixes

```
    def stemming(self):
        st = PorterStemmer()
        self.pr_df["reviews_text_lematized"] = self.pr_df["reviews_text_lematized"].apply(lambda
x: " ".join([st.stem(word) for word in x.split()]))
        return self.pr_df
```

Spelling correction of the reviews

```
    def spelling_correction(self):
        self.pr_df["reviews_text"] = self.pr_df["reviews_text"].apply(lambda x:
str(TextBlob(x).correct()))
        return self.pr_df
```

Algorithm of Linear Support Vector

```
    def linearsvc(self):

        In this line we are rounding the ratings to improve the accuracy of the model.
        self.f_df['reviews_rating'] = self.f_df['reviews_rating'].round()
```

Creates the training and set of the reviews to analyse and the ratings to predict

X_train, X_test, y_train, y_test = train_test_split(self.f_df['reviews_text'], self.f_df['reviews_rating'].astype('int'), test_size = 0.25, random_state=53)

The model consisst of 3 steps:

1. TFIDVectorizer takes the feature and creates a matrix of the bag of words. This algorithm gives every word a certain weight. Words that are very frequent will get a lower rating. The N-gram will tell us how many words to consider, for example 1-2 look separate words but also pair of words.

2. Chi-squared will take out of all the bag of words, we are selecting the best features. In this case the best 8,000 features. This stochastic algorithm gives weighting so the features are very dependent from each other.

3. The las part is the classifier LinearSVC, with lasso penalty.

```
pipeline = Pipeline([
    ('vect', TfidfVectorizer(ngram_range=(1,3), stop_words=None, sublinear_tf=True)),
    ('chi', SelectKBest(chi2, k=8000)),
    ('clf', LinearSVC(C=1.0, penalty='l1',max_iter=3000, dual=False))
            ])
```

With this we have our model ready. The pipeline will pass each of the results thru the pipeline created above.

```
model = pipeline.fit(X_train, y_train)
```

In this part, we get the instances of the pipeline process

```
vectorizer = model.named_steps['vect']
chi = model.named_steps['chi']
clf = model.named_steps['clf']
feature_names = vectorizer.get_feature_names()
feature_names = [feature_names[i] for i in chi.get_support(indices=True)]
feature_names = np.asarray(feature_names)
```

We create the target for the rating

```
target_names = ['1', '2', '3', '4', '5']

print("The top keywords are: ")

# This will print the best keywords for each class we have
for i, label in enumerate(target_names):
    top10 = np.argsort(clf.coef_[i])[-10:]
    print("%s: %s" %(label, " ".join(feature_names[top10])))
```

```
    print("Accuracy score: " + str(model.score(X_test, y_test)))
    print(model.predict(['that was an awesome place. great food!']))

    return model.score(X_test,y_test)
```

`df = clean_text.remove_stop_w()`	It removes stop words from reviews_text
`cwc = clean_text.count_rare_word()`	Count the rare words in the reviews.
`df = clean_text.remove_rare_words()`	This will clean the rare words from the reviews column
`#df = clean_text.spelling_correction()`	This will do the spelling correction
`df = clean_text.tokenization()`	Tokenization: Convert to strings
`df = clean_text.lematization()`	Converts the word into its root word
`df = clean_text.stemming()`	This will do the stemming process

```
This the main page. It will show the links to the other pages
index_page = html.Div([
    html.H2('Hotel Reviews - Text Mining', style={'textAlign': 'left', 'color': '#0f2128'}),
    dcc.Link('Upload Data', href='/page-1'),
    html.Br(),
    dcc.Link('Preprocessing', href='/page-2'),
    html.Br(),
    dcc.Link('Histogram', href='/page-3'),
    html.Br(),
    dcc.Link('Word Map', href='/page-4'),
])


#Renzo: This is the layout of the 1st page. Upload the csv file
page_1_layout = html.Div([
    html.H2('Hotel Reviews - Upload your file', style={'textAlign': 'left', 'color': '#0f2128'}),
    dcc.Upload(
        id='upload-data',
        children=html.Div([
            'Drag and Drop or ',
            html.A('Select Files')
        ]),
```

Redirect to each of the pages

```
def display_page(pathname):
    if pathname == '/page-1':
        return page_1_layout
    elif pathname == '/page-2':
        return page_2_layout
    elif pathname == '/page-3':
        return page_3_layout
    else:
        return index_page
```

4. Results

From the results with Linear Vector Machine and LASSO regression, the model has an accuracy of 50%. Considering that the dataset has only 10,000 rows, we would expect that the model will improve its accuracy if more data was provided.

To achieve this result, as there is no specific formula for all the cases, some parameters were adjusted. In the first case of the test size, the values from 20% to 35% were tried; which end up being 25% that size of the test that will give the higher accuracy.

In the case of the n-grams, using 1-1, 1-2 wouldn't give better results, which end up being 1-3 N-grams the optimal number. The K-value for the chi-squared it was also tested with different numbers ranging from 6000 to 12000. The number that best results had was k=8000.

Below is an example of the results obtained by the model. As we can see, the model is classifying the reviews to each rating. The reviews are ordered by the number of rating, and it's being extracted as a result of the algorithm.

1: dump rude staff disgusting room people refused need repair breakfast front room immediately worst pest

2: service staffs micro reset bad cleanliness everything old needs major made difficult go dinner falling apart complained

3: sparse unacceptable disorganized incompetent pillow case end night show age lot desired today better places

4: ac okay door bad room great hotel bad pool however good hotel bad valet nice price

5: great experience lovely thank cant wait love hotel amazing excellent exceptional loved hotel one best

The accuracy score of this model is 50%.

5. Summary and conclusions

The project aims to answers the following question: Does the customers' reviews can be useful to predict the rating of the hotels?

Applying different data mining techniques, it can be concluded that it is possible to predict the rating of a hotel based on the reviews.

It is necessary to apply different data cleaning and preprocessing (feature depuration, noise reduction, remove unnecessary words, special characters) as well as performing advance techniques like stemming or lemmatization before applying the model,

In case of the linear vector machine with lasso penalty, the model could predict a 50%. If we consider that the data it is only 10,000 instances, we would expect that the model increase the percentage of prediction with more data. We can conclude that the model is better with 1-3 N-grams, compared to 1-1 or 1-2. We have also improved this model by using chi-squared to improve the feature selection.

In conclusion, with a dataset of 10,000 instances, it was possible to predict a 50% with the Linear vector machine.

6. Percentage of code

$$\frac{\mathbf{85-37}}{\mathbf{85-30}}\boldsymbol{X}\mathbf{100} = \mathbf{87.27\%}$$

7. References

R. (n.d.). *An Idiot's guide to Support vector machines (SVMs)*. Lecture.

Text Analytics for Beginners using NLTK. (n.d.). Retrieved from https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk

Tibshirani, R. (1996) Regression shrinkage and selection via the lasso. J.R.S.S.B. 58, 267-288

A. (2016, February 24). *Topics in sparse Support Vector Machines*. University of Arizona.

https://www.math.arizona.edu/~wammonj/talks/topics-sparse-svms.pdf

J., S., T., & R. (n.d.). *1-norm Support Vector Machines*. Stanford University

https://papers.nips.cc/paper/2450-1-norm-support-vector-machines.pdf