Amsterdam University of Applied Sciences

SENTIMENT ANALYSIS ON HOTEL REVIEWS

Research Report

Data Engineer & Data Scientist
2019 – 2020

SENTIMENT ANALYSIS ON HOTEL REVIEWS

Research Report

AUTHOR
Yoshio Schermer (500760587)

COURSE
Data Engineer & Data Scientist

DATE
3/31/2020

TYPE PROJECT
Research Report

VERSION
1.0

Summary

This report describes sentiment analysis on hotel reviews for determining whether a review is positive or negative. The dataset used contains more than 500.000 labeled hotel reviews. To perform sentiment analysis the following supervised machine learning algorithms were used: Multinomial Naïve Bayes, Bernoulli Naïve Bayes, Linear Support Vector Classifier, and Logistic Regression.

Grid search cross validation (5-fold) was used to tune text feature extraction parameters and hyperparameters of the algorithms.

The result was that Linear Support Vector Classifier using 10.000 features, bigrams and IDF set to false performed highest with an accuracy of 94.87%. Second to this, was Logistic Regression using 10.000 features, bigrams and IDF set to true with an accuracy of 94.85%. Logistic Regression has a slightly higher F1-Score than the Linear Support Vector Classifier.

The code for this project can be found at:
https://github.com/yschermer/sentiment_analysis_hotel_reviews

Table of Contents

1. Introduction

This research report describes various supervised machine learning algorithms and the process of using them in sentiment analysis.

The use case is as follows: build a classifier that can accurately classify a hotel review on its polarity, so whether it is a positive review or a negative review. For this we use a dataset of more than 500.000 hotel reviews, which can be found at https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe.

The report follows the outline: background, methods, results, and conclusions. In the first chapter, we'll describe various supervised machine learning algorithms that are typically used in sentiment analysis.
In the subsequent chapter, we go through data discovery, data preparation, and model building using the algorithms from the previous chapter.
In the chapter on results, we plot and compare the performance classifiers.
Finally, in conclusions, we discuss the results and provide new research questions following this.

2. Background

2.1 Introduction

There are many supervised machine learning algorithms. But there are only a few that perform exceptionally well for the task of sentiment analysis. These are Naïve Bayes and Support Vector Machines (Ismail, 2016) (Wang, 2012).

2.2 Naïve Bayes

Naïve Bayes is a probabilistic classifier. The formula of it can be seen in Figure 1.

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \rightarrow Posterior = \frac{likelihood \times prior}{evidence}$$

Figure 1. Bayes Theorem

Notable in the formula is the conditional probability $P(X|Y)$. As X grows, the necessary computations grow exponentially, as can be seen in Figure 2.

$$P(X = (x_1, x_2, \dots, x_k)|Y = y) \; is \; called \; a \; parameter \; in \; Naive \; Bayes.$$

$$If \; x_j \; \forall j \; is \; binary, there \; are \; 2^{k+1} - 1 \; parameters$$

Figure 2. Exponential growth of parameters when conditionally dependent.

To avoid these expensive computations, we make the naïve assumption that all the features are conditionally independent (see Figure 3). Thereby significantly reducing the parameters, which now grows linearly instead of exponentially (see Figure 4).

$$P(Y|X) = \frac{P(Y)\prod_i P(X_i|Y)}{P(X)}$$

Figure 3. Naïve assumption of conditional independence.

$$If \; x_j \; \forall \; is \; binary, there \; are \; 2k \; parameters$$

Figure 4. Linear growth of parameters when naïve assumption of conditional independence holds.

2.3 Support Vector Machine

Like the Naïve Bayes classifier, support-vector machines (also known as SVMs) is also a supervised machine learning algorithm. An SVM in contrast to Naïve Bayes, is a non-probabilistic binary linear classifier (Support-vector machine, 2020).

Given data points belonging to either one of two classes, an SVM maps these on a plane and tries to find a hyperplane that represents the largest separation / margin between the two classes (see Figure 5). This margin can be increased to avoid overfitting, but as a result does accept outliers to belong to the wrong class (Support-vector machine, 2020).

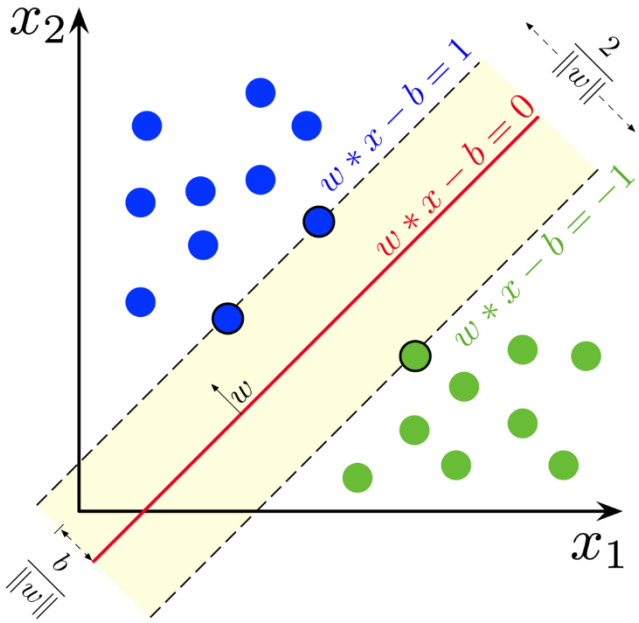

Figure 5. Maximum-margin hyperplane and margin for an SVM trained on two classes. Samples on margins are called support vectors (Larhmam, 2018).

2.4 Logistic Regression

While linear regression only deals with continuous variables, logistic regression deals with binary classes. Logistic regression is linear regression but transformed with a sigmoid function so the predictive values only fall between 0 and 1 (Liu, 2020).

As can be seen in Figure 6, logistic regression tries to create a sigmoid function that best fits the data points with its probability and dependent variable. In the same figure, we see that the function outputs a higher probability given more hours of studying.

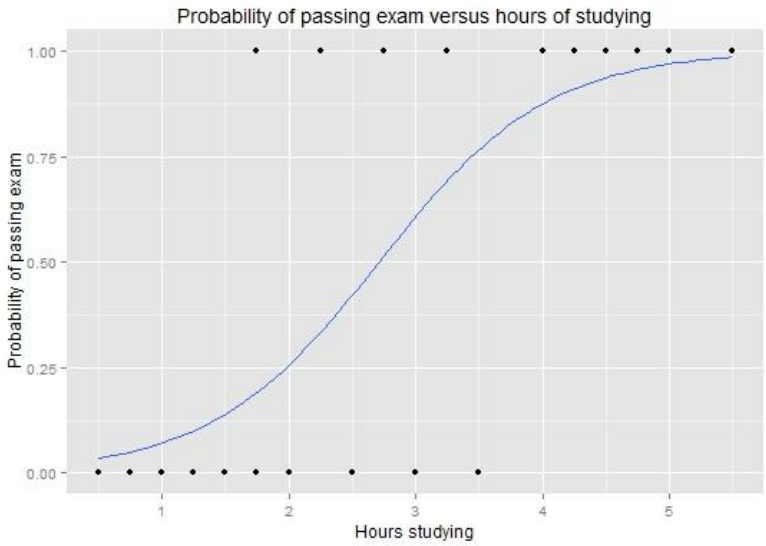

Figure 6. Graph of a logistic regression curve showing probability of passing an exam versus hours studying (G, 2015)

3. Methods

3.1 Data Discovery

The dataset that was used for training the classifiers consists of more than 500.000 hotel reviews scraped from Booking.com. The dataset can be found at the following link: https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe.

Besides this, a handwritten dataset of 10 labeled reviews was added to this. See Appendix B.

3.2 Data Preparation

3.2.1 Importing

The dataset (which is in .csv format) was imported to MySQL as a single table. Subsequently, this table was queried for further processing.

3.2.2 Fetching

The stored procedures in Table 1 were made to accommodate further processing.

Stored Procedure	Description	Input
SelectReviews	Selects all the reviews with the following columns: - Negative review - Negative review word count - Positive review - Positive review word count	
SelectTopReviews	Selects X reviews starting from the top with the following columns: - Negative review - Negative review word count - Positive review - Positive review word count	limit_from_top (int)
SelectReviewsInRange	Selects reviews between given indices with the following columns: - Negative review - Negative review word count - Positive review - Positive review word count	begin_index (int), end_index (int)

Table 1. All stored procedures available for fetching.

The stored procedures for selecting just a subset of all the reviews was useful for testing whether pipelines and models were correctly programmed.

3.2.3 Pre-processing and cleaning

After having fetched the reviews, the first thing I did was placing all of them in a Pandas DataFrame.

Subsequently, I split the positive reviews from the negative reviews. I noticed that using a copy-and-drop strategy significantly reduced the time to do this as can been seen in Figure 7.

```
# splitting dataframe into negative and positive
# using np.split: 20+ seconds
# using copy and drop: <1  second
dfs = df.copy(deep=True)
negative_reviews = df.drop(columns=['positive_review', "positive_word_count"])
positive_reviews = dfs.drop(columns=['negative_review', "negative_word_count"])
```

Figure 7. Copy-and-drop strategy for splitting reviews.

When looking through the dataset before, I noticed a lot of bogus reviews, containing texts such as: “Nothing”, “No Negative”, “n a”, and so on. Since, there were many of these both in the positive as the negative reviews, it will add unnecessary noise to the dataset. That’s why I made a list of these bogus reviews and removed them from both DataFrames as can been seen in Figure 8.

```
# remove bogus or empty reviews
bogus_reviews = ["NOTHING", "Nothing", "nothing", "Nothing at all",
                 "No Negative", "No Positive", "n a", "N a", "N A", " "]
negative_reviews = negative_reviews[negative_reviews.isin(
    {"review": bogus_reviews})['review'] == False]
positive_reviews = positive_reviews[positive_reviews.isin(
    {"review": bogus_reviews})['review'] == False]
```

Figure 8. Removing irrelevant reviews.

Since the DataFrames contain a word count describing the positivity or the negativity of the review, we could use this given information to normalize and label the DataFrames for positive or negative review. This is being done in Figure 9.

```
# normalize negative reviews
negative_threshold = 0
negative_labels = (negative_reviews['word_count']
                   > negative_threshold).astype(int)
negative_reviews['label'] = negative_labels

# remove non-negative reviews
negative_reviews = negative_reviews[negative_reviews['label'] == 1]
```

Figure 9. Normalizing and labelling negative reviews.

Of course, after having done the same for the DataFrame with positive reviews, we’d end up having two DataFrames with a label value of 1. So, the label of the DataFrame containing negative reviews, was

turned to -1. This way both DataFrames could be merged without information loss. This can be seen in Figure 10.

```
# A negative review will have as label value: -1.
# A positive review will have as label value: 1.
negative_reviews = negative_reviews.assign(label=-1)

# merging reviews and labels
reviews = pd.concat([negative_reviews, positive_reviews],
                    join="inner", ignore_index=True)
```

Figure 10. Re-assigning negative review label value and merging reviews.

Finally, we observe the characteristics of the DataFrames. The findings are displayed in Table 2

Amount of negative reviews:	386999
Amount of positive reviews:	479609
Predicting only 1 (positive review):	55.34% accuracy

Table 2. Observing characteristics of hotel reviews.

3.3 Model Building

For model building I had made a pipeline that does text feature extraction (that is, vectorization) and then trains a classifier. This can be seen in Figure 11. In the same figure, you will see **`model['object']`**, which I will explain next.

```
text_clf = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('clf', model['object'])])
```

Figure 11. Pipeline that does text feature extraction and trains a classifier.

Since I want to train multiple classifiers using the pipeline I made, I created a dictionary that contains a model, parameters for text feature extraction, and hyper parameters for the model (see Figure 12).

```
models = [
    {
        "id": "mnb",
        "name": "Multinomial Naive Bayes",
        "object": MultinomialNB(),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
            'tfidf__use_idf': (True, False),
        }
    },
    {
        "id": "bnb",
        ...
```

Figure 12. First model in dictionary containing all classifiers to be trained by the pipeline. The entire dictionary containing all models can be found in *Appendix A*.

When training of the classifier starts in the pipeline, I use **grid search cross validation** for finding the best parameters and estimator. Also, I try to avoid *overfitting* the training data by doing cross-fold validation (5 splits) as can be seen in Figure 13. The grid search cross validation tries to find the best parameters and estimator for **accuracy**. The reason for this is because (1) we have a relatively balanced dataset making this possible, and (2) in this case we want to find as many true positives as true negatives, we care less about misclassification.

```
    gs_clf = GridSearchCV(
        text_clf, model['parameters'], scoring='accuracy', cv=5, n_jobs=1)
```

Figure 13. Grid search cross validation with 5 folds and estimating for accuracy.

The parameters given to the grid search cross validation are described in the previously mentioned dictionary (see Figure 12). Each classifier is, however, trained with the same set of parameters for text feature extraction, which can be seen in Table 3.

Parameter	Values
Max features	1000, 10000
Stop words	English, None
N-gram range	Uni-gram, bi-gram
Use IDF (weighs down unimportant terms)	True, False

Table 3. Parameters text feature extraction.

Besides the parameters for text feature extraction, there are a few hyper parameters that I also take into the grid search cross validation. In particular, the Linear Support Vector Classifier has a hyperparameter, C. This hyperparameter controls the size of the soft margin (see Figure 14). A lower C results in more bias and less variance, while a higher C results in less bias and higher variance (Greene, 2016). The default value is 1, but I choose to test 0.1 for C as well, since I wanted to test if accepting more outliers results in an improved estimator. All the results are displayed in the next chapter.

```
    {
        "id": "lsvc",
        "name": "Linear Support Vector Classifier",
        "object": LinearSVC(),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
            'tfidf__use_idf': (True, False),
            'clf__C': [0.1, 1],
        }
    },
```

Figure 14. Linear Support Vector Classifier with hyperparameter C.

After having trained and tested, I would plot/print the classification metrics. And finally, I would save the models using *pickle* as is shown in Figure 15.

```
    # save model
    print("Saving classifier...")
    filename = '{}.sav'.format(model['id'])
    pickle.dump(gs_clf, open(filename, 'wb'))
    print("Classifier successfully saved in: {}.sav\n".format(model['id']))
```

Figure 15. Saving the model using pickle.

4. Results

From grid search cross validation, I got the best estimators with the parameters provided. These estimators were then tested and the performance of them were evaluated. In Table 4, we see the performance of all these estimators/classifiers side-by-side.

Classifier	Accuracy	Precision	Recall	F1-Score	Fit Time (sec)
Multinomial Naïve Bayes	92.79	neg = 0.9114 pos = 0.9410	neg = 0.9278 pos = 0.9274	neg = 0.9195 pos = 0.9341	66.7
Bernoulli Naïve Bayes	90.97	neg = 0.8740 pos = 0.9399	neg = 0.9292 pos = 0.8921	neg = 0.9007 pos = 0.9154	51.4
Linear Support Vector Classifier	**94.87**	**neg = 0.9332** pos = 0.9606	neg = 0.9521 **pos = 0.9449**	neg = 0.9425 pos = 0.9527	**49.3**
Logistic Regression	94.85	neg = 0.9331 **pos = 0.9613**	**neg = 0.9529** pos = 0.9448	**neg = 0.9429** **pos = 0.9530**	53.9

Table 4. Performance evaluation and comparison among classifiers. Scores in bold are the highest for that metric.

From this, we can see that Linear Support Vector Classifier reached the highest accuracy by a small margin over Logistic Regression and has the lowest training time. Interesting, however, is that Logistic Regression has the highest F1-Score indicating that it is slightly more precise and has higher recall. The associated confusion matrices can be seen in Figure 16.

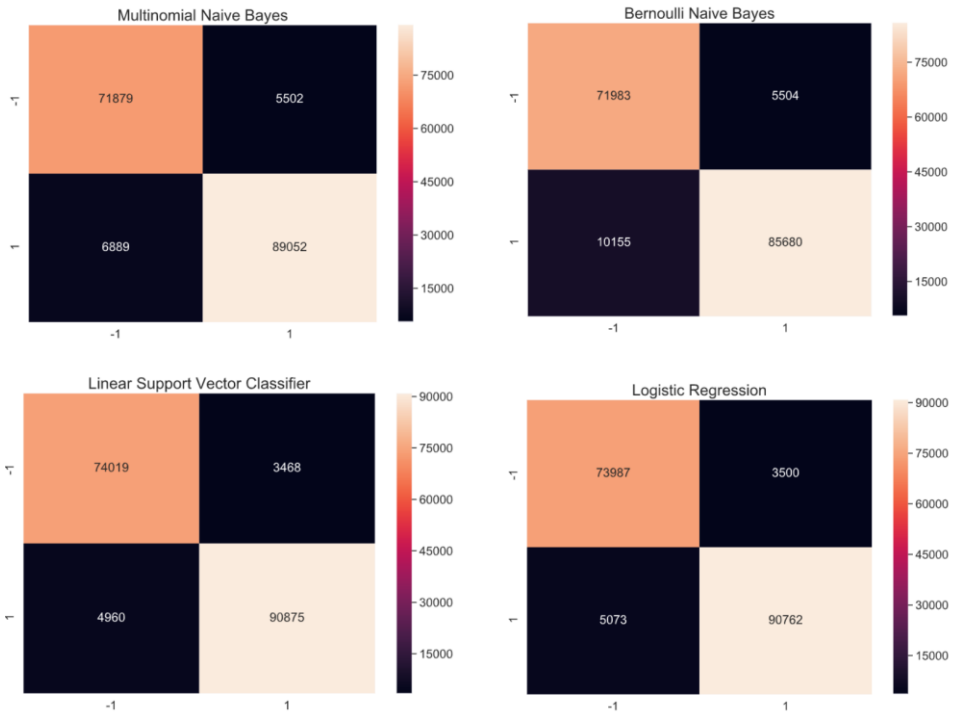

Figure 16. Confusion matrices per classifier (left column = predicted negative, right column = predicted positive, upper row = actual negative, lower row = actual positive).

The parameters that belong to the best estimators, hence the best parameters are shown in Table 5.

Classifier	Max features	Stop words	N-gram	Use IDF	C	Max iteration
Multinomial Naïve Bayes	10.000	None	Bi-gram	True	n/a	n/a
Bernoulli Naïve Bayes	10.000	English	Bi-gram	True	n/a	n/a
Linear Support Vector Classifier	10.000	None	Bi-gram	False	1	n/a
Logistic Regression	10.000	None	Bi-gram	True	n/a	10.000

Table 5. Best parameters of best estimators.

In (Wang, 2012) it was assessed that bi-grams typically outperform unigrams in sentiment analysis. The same result I find when training my own classifiers for this task.

5. Conclusion

We have researched various supervised machine learning algorithms and seen the process from data discovery until model building and evaluation using these algorithms. We have used throughout the process grid search cross validation to avoid overfitting and simultaneously tune text feature extraction parameters and hyperparameters of our models.

As a result, we found that Linear Support Vector Classifier with default hyperparameters and Logistic Regression with max iteration equal to 10.000 performed best. See the parameters for text feature extraction in Table 5. On the accuracy metric, they scored 94.87% and 94.85% respectively. However, Logistic Regression scores slightly higher on F1.

Interesting would be to try different hyperparameter values for max iteration and Logistic Regression. Also, since Support Vector Machines are known to do well on sentiment analysis (Ismail, 2016), it would be interesting to try Support Vector Classifier with a kernel other than linear.

6. Bibliography

G, M. (2015). *Exam pass logistic curve.* Retrieved from Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Exam_pass_logistic_curve.jpeg

Greene, C. (2016). *The C Parameter for Support Vector Machines*. Retrieved from Youtube: https://www.youtube.com/watch?v=5oVQBF_p6kY

Ismail, H. &. (2016). A Comparative Analysis of Machine Learning Classifiers for Twitter Sentiment Analysis.

Larhmam. (2018). *SVM margin.* Retrieved from Wikimedia: https://commons.wikimedia.org/wiki/File:SVM_margin.png

Liu, C. (2020). *Linear - Logistic Regression Explained.* Retrieved from kdnuggets.com: https://www.kdnuggets.com/2020/03/linear-logistic-regression-explained.html

Support-vector machine. (2020). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Support-vector_machine

Wang, S. a. (2012). Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. *Association for Computational Linguistics*.

7. Appendix A: Models

```
models = [
    {
        "id": "mnb",
        "name": "Multinomial Naive Bayes",
        "object": MultinomialNB(),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
            'tfidf__use_idf': (True, False),
        }
    },
    {
        "id": "bnb",
        "name": "Bernoulli Naive Bayes",
        "object": BernoulliNB(),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
            'tfidf__use_idf': (True, False),
        }
    },
    {
        "id": "lsvc",
        "name": "Linear Support Vector Classifier",
        "object": LinearSVC(),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
            'tfidf__use_idf': (True, False),
            'clf__C': [0.1, 1],
        }
    },
    {
        "id": "lr",
        "name": "Logistic Regression",
        "object": LogisticRegression(solver="lbfgs"),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
```

```
            'tfidf__use_idf': (True, False),
            'clf__max_iter': [10000]
        }
    },
    {
        "id": "knn",
        "name": "K-Nearest Neighbors Classifier",
        "object": KNeighborsClassifier(),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
            'tfidf__use_idf': (True, False),
        }
    },
    {
        "id": "gradient_boost",
        "name": "Gradient Boosting Classifier",
        "object": GradientBoostingClassifier(),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
            'tfidf__use_idf': (True, False),
        }
    },
    {
        "id": "ada_boost",
        "name": "AdaBoost Classifier",
        "object": AdaBoostClassifier(),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
            'tfidf__use_idf': (True, False),
        }
    },
    {
        "id": "tree",
        "name": "Decision Tree Classifier",
        "object": DecisionTreeClassifier(),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
            'tfidf__use_idf': (True, False),
        }
```

```
    },
    {
        "id": "rfc",
        "name": "Random Forest Classifier",
        "object": RandomForestClassifier(n_estimators=100),
        "parameters": {
            'vect__max_features': (1000, 10000),
            'vect__stop_words': ["english", None],
            'vect__ngram_range': [(1, 1), (1, 2)],
            'tfidf__use_idf': (True, False),
        }
    },
]
```

8. Appendix B: Handwritten reviews

```
review, label
 This rooms smells wonderfully awful., -1
 The view was amazing., 1
 Windows are so dirty., -1
 Great bed, 1
 Didn't get my money back for checking out one day earlier., -1
 The swimming pool was kinda cool., 1
 It wasn't very comfortable sitting on the wooden chairs, -1
 I could see Mt. Fuji from the apartment!, 1
 This was the worst hotel ever. I couldn't even sleep one night, -1
 The cookies were tasty., 1
```