YSC2244: Programming for Data Science
Assignment 3 - Sentiment prediction with Machine Learning
By Raihanah Nabilah Fatinah (A0207754B)

Table of Content

1. General overview of the task and the report

In this assignment, we are asked to load the movie reviews data, train the data, and then we should predict that given the film review, whether or not the film review is a positive review or a negative review. In doing this, we are combining our knowledge in text mining and machine learning.

In this assignment, I tried different methods in order to write my sentiment prediction.

1. Method 1: I use the package *textblob* where I use a ready-to-execute function to predict a text without training the data myself.
2. Method 2: I use the package nltk where I train the data myself and build the model and use the model to predict a text. I write my own feature_extraction function here.
3. Method 3: Similar to method 2, I use the package nltk where I train the data myself and build the model and use the model to predict a text. The difference with method 2 is that here, I use the in-built feature_extraction function from sklearn to save computing time.

2. Understanding the machine learning framework

According to Professor Bodin's lecture, there are two types of Machine Learning: supervised and unsupervised systems. Our assignment is a supervised learning since we train the model to contain expected outputs, which is either positive review or negative review.

The framework that is used by the supervised machine learning is shown in the following diagram:

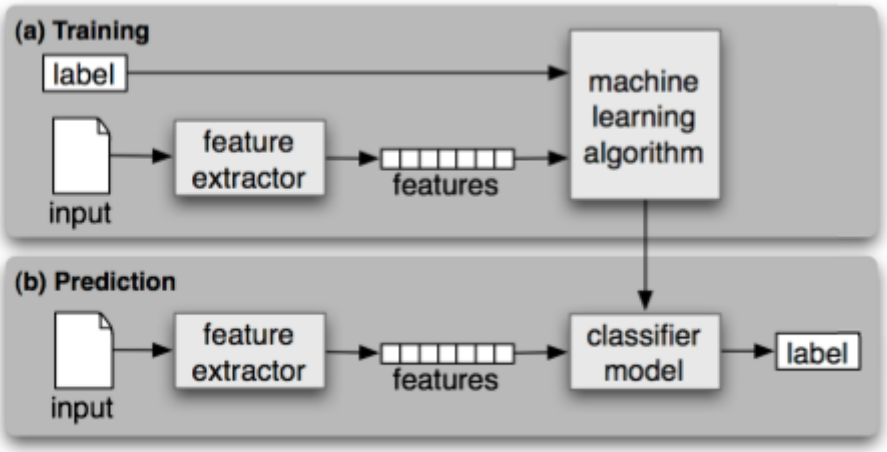


Figure 1.1: Supervised machine learning framework[1]

When we are using machine learning to do sentiment analysis, we first need to build our classifier model. To build the model, we need to train our data using a machine learning algorithm. In this training process, we are essentially teaching the computer how to identify whether a movie review is positive or negative.

[1] NLTK, "Learning to Classify Text", https://www.nltk.org/book/ch06.html

During training, we first prepared our input data that we want to train. Then, we created our feature vector. Then, we created a function called feature extractor whereby the function is used to convert each input value to a feature set.[2] Then, these feature sets captured the basic information about each input that will be used to classify our input.[3] Then, we fed the pairs of feature sets and labels to the machine learning algorithm to build a model.[4]

Once we have built our model, we use this model to predict the sentiment of our data. Similar to the process in training, during the prediction, we use the same feature extractor function to convert our input data into feature sets. Then, we fed the feature sets into the model, which then will generate the predicted label or sentiment of our data.

During the prediction, we should use the data that is different from the data that we trained. If we used the same data to test our model, then the accuracy will be 100% because we taught the computer with that exact data.[5]

3. Method 1: Using ready-to-execute package, *textblob*

3.1 The Predict sentiment function

Initially, I did not fully understand how I could build my own model from scratch. This is why I ended up using packages like *textblob*, whereby each of these packages have an in-built function that automatically built the model and executed the data training. Hence, using this function, all I have to do is to create a predict sentiment function to predict the sentiment of my input data. The following is my predict_sentiment function using textblob:

```
from textblob import TextBlob

def predict_sentiment(text):
    nlp = TextBlob(text).sentiment.polarity
    if nlp > 0:
        return "pos"
    else:
        return "neg"
```

Figure 2.1: My *predict_sentiment* function using *textblob* package

I consulted Professor Bodin on this, and he mentioned that I could do this method; however, I need to understand how the textblob package built their model, mainly on what data the package is using and how they trained the data.

3.2 Understanding how the package train its data

I did some research into *textblob* reading docs, and I found the following process.

---

[2] Ibid.
[3] Ibid.
[4] Ibid.
[5] Ibid.

```
@requires_nltk_corpus                                                    [docs]
def train(self):
    """Train the Naive Bayes classifier on the movie review corpus."""
    super(NaiveBayesAnalyzer, self).train()
    neg_ids = nltk.corpus.movie_reviews.fileids('neg')
    pos_ids = nltk.corpus.movie_reviews.fileids('pos')
    neg_feats = [(self.feature_extractor(
        nltk.corpus.movie_reviews.words(fileids=[f])), 'neg') for f in neg_ids]
    pos_feats = [(self.feature_extractor(
        nltk.corpus.movie_reviews.words(fileids=[f])), 'pos') for f in pos_ids]
    train_data = neg_feats + pos_feats
    self._classifier = nltk.classify.NaiveBayesClassifier.train(train_data)
```

Figure 2.2: The *train* function using *textblob* package[6]

```
def _default_feature_extractor(words):
    """Default feature extractor for the NaiveBayesAnalyzer."""
    return dict(((word, True) for word in words))
```

Figure 2.3: The *default_feature_extractor* function using *textblob* package[7]

```
def __init__(self, feature_extractor=_default_feature_extractor):
    super(NaiveBayesAnalyzer, self).__init__()
    self._classifier = None
    self.feature_extractor = feature_extractor
```

Figure 2.4: The *feature_extractor* is automatically using the *default_feature_extractor* in the textblob package if not provided the *feature_extractor* function[8]

To train the data, the *textblob* package executes the train function, where they mainly use the data from the *nltk* movie reviews package, as seen in figure 3. Then, they separated the movie reviews file list that are categorized as negative and positive. Then, they used the default_*feature_extractor* function where they will return True if the word is in the movie reviews data, as seen in figure 4 and 5, thereby creating our feature set, seen in figure 3 written as *train_data* = *neg_feats* + *pos_feats*. Then, the feature set is fed to the machine learning algorithm, and in this case, it is the Naive Bayes classifier, where it built our classifier model.

I found that, using this method, the accuracy that I achieved is around 70.3%. The python code of this method is provided in the index.

---

[6] TextBlob, "TextBlob Sentiment Documentation", https://textblob.readthedocs.io/en/dev/_modules/textblob/en/sentiments.html#NaiveBayesAnalyzer.train

[7] Ibid.

[8] Ibid.

Figure 2.5: Result using *textblob* sentiment package

Since *textblob* was built highly on the nltk package, I got interested in building my own model using the *nltk* package as well. This leads me to my second method below.

4. Method 2: Using *nltk, with building my own feature extraction function*

My understanding in building this sentiment analysis using the *nltk* package is largely influenced by online resources that I found.[9] The data that will be used to build the model is the *nltk*'s movie review data.

```
import nltk
from nltk.corpus import movie_reviews
```

Figure 3.1: Importing the *nltk* package and the *movie_reviews* data

4.1 Building the feature vector and processing the data

First, we need to create the feature vector, whereby we take the first 4000 words of the most frequent words in the *nltk*'s movie review data. We don't include all the words in the movie reviews in order to save power and time. Hence, we are compromising on the result that we will get.

```
all_words = nltk.FreqDist(movie_reviews.words())
feature_vector = list(all_words)[:4000]
```

Figure 3.2: Creating the feature vector

However, this list of words or the feature vector is not processed properly. To properly process this, we remove the list of stop words using the *nltk stop_words* package and also the punctuations using the word.isalpha()

```
from nltk.corpus import stopwords
nltk.download('stopwords')
stopWords = set(stopwords.words("english"))
filtered_words = [x for x in all_words if x not in stopWords]
up_words = [word for word in filtered_words if word.isalpha()]
```

Figure 3.3: Removing the stop words and the punctuations

Then, we updated our feature_vector with the processed words without stop words and punctuations.

[9] Franklin, "Sentiment Analysis in NLTK Python", https://medium.com/@joel_34096/sentiment-analysis-of-movie-reviews-in-nltk-python-4af4b76a6f3

```
feature_vector = list(up_words)[:4000]
```

Figure 3.4: Updating the feature_vector

Then, we created a list of documents, where it contains the list of words of each movie review and the category label of each review (positive or negative).

```
document = [(movie_reviews.words(file_id),category)
            for file_id in movie_reviews.fileids()
            for category in movie_reviews.categories(file_id)]
```

Figure 3.5: Creating the word-list and labelling each movie review

4.2 Training the data, building the model

Then, we wrote our feature extractor function that checked whether the words in the *feature_vector* are present in the given document. If the word in the feature_vector is also in the movie review (or the document), it will return True. Else, it will return False.

```
def feature_extraction(wordlist):
    feature = {}
    for word in feature_vector:
        feature[word] = word in wordlist
    return feature
```

Figure 3.6: Our feature extraction function

Then, we apply our feature extractor function to our document, and we named our list of features that contains the feature and the category of each movie review as *feature_sets*.

```
feature_sets = [(feature_extraction(wordlist),category)
                for (wordlist,category) in doc]
```

Figure 3.7: Our feature sets

Once we have our feature sets, we separate our train set and test set to 75:25 ratio. Remember that the train set and test set cannot be the same data set. We then trained our data using the SVC algorithm in the SklearnClassifier package.

```
train_set, test_set = model_selection.train_test_split(feature_sets,test_size = 0.25)
model = SklearnClassifier(SVC(kernel = 'linear'))
model.train(train_set)
```

Figure 3.8: Training the model using SVC SklearnClassifier

I then tested the accuracy of this method, and I found that the accuracy is around 81.4%.

```
1 accuracy = nltk.classify.accuracy(model,test_set)
2 accuracy*100
```

```
81.39999999999999
```

Figure 3.9: Accuracy of the SVC SklearnClassifier

4.3 Deciding the best model

I then researched further online, and there are many algorithms that we can train our data with.[10] For Naive Bayes methods, there are MultinomialNB and BernoulliNB algorithms. For the Linear Model method, there are Logistic Regression and SGDClassifier algorithms. For SVM methods, there are SVC, LinearSVC, and NuSVC algorithms. I then built the model and trained my data with all these algorithms. Then, I tested the accuracy of each of these model, and I got the following result:

```
SVC accuracy percent: 81.39999999999999
LogisticRegression_classifier accuracy percent: 84.6
MNB_classifier accuracy percent: 82.39999999999999
BernoulliNB_classifier accuracy percent: 80.60000000000001
SGDClassifier_classifier accuracy percent: 82.6
LinearSVC_classifier accuracy percent: 81.39999999999999
NuSVC_classifier accuracy percent: 84.2
```

Figure 3.10: Accuracy of different models

From figure 16, it is clear that the Logistic Regression classifier has the highest accuracy level, which is 84.6%. Hence, I decided to go ahead with my logistic regression model.

4.4 The predict sentiment function

Then, I wrote my *predict_sentiment* function:

```
def predict_sentiment(text):
    words = word_tokenize(text)
    words_filtered = [word for word in words if word.isalpha()]
    feature = feature_extraction(words_filtered)
    res = model.classify(feature)
    return res
```

Figure 3.11: Predict sentiment function

Once I had my predict sentiment function, I tried to test my function by running tester.py. However, it took a very long time for this function to run because training the data alone and creating the *feature_sets* by using my own feature extraction function takes a significant amount of time (approximately 2.5 hours), exceeding the upper time limit given by Prof which is 2 hours. The python notebook and code of this method is provided in the index.

With this, I decided to do another method, which is the method that was taught by Professor Bodin in the class and during office hours.

[10] Python Programming, “Sklearn Scikit Tutorial”

5. Method 3: Using *nltk*, with using in-built feature extraction function

5.1 Data processing and cleaning

In this method, I first followed the data processing and data cleaning that we learned in Week 10's Text Mining.

We first processed the movie reviews data that we want to train and turning it into a data frame, using the load_reviews() function that we wrote in Week 10. I named our dataframe as *data*.

```
def load_reviews():
    keys = ['filename','kind','text']
    res = {}

    for i in keys:
        res[i] = []

    neg_files = movie_reviews.fileids('neg')
    for file in neg_files:
        res['filename'].append(os.path.basename(file))
        res['kind'].append('neg')
        res['text'].append(movie_reviews.raw(file))

    pos_files = movie_reviews.fileids('pos')
    for file in pos_files:
        res['filename'].append(os.path.basename(file))
        res['kind'].append('pos')
        res['text'].append(movie_reviews.raw(file))

    return pd.DataFrame.from_dict(res)

data = load_reviews()
```

Figure 4.1: *load_reviews* function to load our movie_reviews data and turn it into a dataframe

Then, we tokenized the text review into the list of words, filtered the words to remove stop words and non-words characters, and lemmatized the words so that we can limit our dictionary to save computational time. To do this, we created a function called *tokenize_filter_and_lemmatize*. Then, once we have done with data cleaning and lemmatizing, we merged the words list into one text again for each review.

```
def tokenize_filter_and_lemmatize(text):
    word = word_tokenize(text)
    filter_1 = [w for w in word if w.isalpha()]
    filter_2 = [w for w in filter_1 if w not in stopWords]
    lemmatize = [wordnet_lemmatizer.lemmatize(w) for w in filter_2]
    return lemmatize

data['text'] = data['text'].apply(lambda x: tokenize_filter_and_lemmatize(x))

data['text'] = data.text.apply(lambda x: ' '.join(x))
```

Figure 4.2: *tokenize_filter_and_lemmatize* function and merging the word list into one text again for each review

5.2 Training the data, building our model

Then, we prepare our data features. To do this, we follow the method that we learned in Week 9's Machine Learning class, where we use a feature extractor function from sklearn

called TF-IDF, or in other words, we use frequency-inverse document frequency, to classify our input data. We first have to convert our data frame review text into an array list, named as X, and we applied TF-IDF into X with fit_transform.

```
X = data.text.to_numpy()
Y = data.kind.to_numpy()

from sklearn.feature_extraction.text import TfidfVectorizer
vec = TfidfVectorizer()
X = vec.fit_transform(X)
```

Figure 4.3: Feature extraction function with TF-IDF

Then, we build our model. Similar to what I did in Method 2, I built the model with multiple algorithms, such as MultinomialNB and BernoulliNB algorithms for Naive Bayes methods, Logistic Regression and SGDClassifier algorithms for Linear Model methods, and SVC, LinearSVC, and NuSVC algorithms for SVC methods. I will test the accuracy of each of these models later after I wrote my predict_sentiment function to determine which one is best to use.

```
model = MultinomialNB()
model.fit(X, Y)

model = BernoulliNB()
model.fit(X, Y)

model = SGDClassifier()
model.fit(X, Y)

model = LogisticRegression()
model.fit(X,Y)

model = SVC()
model.fit(X,Y)

model = LinearSVC()
model.fit(X, Y)

model = NuSVC()
model.fit(X,Y)
```

Figure 4.4: Different models to train the data

5.3 The predict sentiment function

Then, I wrote my predict_sentiment function, which takes the text input, splits the text into list of words, filters the stopwords and the non-word characters, lemmatize the words, merges the list of words back into a text, then predicts the result using our model.

```
def predict_sentiment(text):
    words = tokenize_filter_and_lemmatize(text)
    review = [" ".join(words)]
    test = vec.transform(review)
    predict = model.predict(test)[0]
    return predict
```

Figure 4.5: My *predict_sentiment* function

5.4 Deciding the best model

Then, I tested the accuracy of each of the models in figure 4.4 by running my *predict_sentiment* function through tester.py, and I got the following result:

Table 1: Accuracy of different models. See index for the screenshot of accuracy result of each model.

Methods	**Model**	**Accuracy**
Naive Bayes	MultinomialNB	80.7%
	BernoulliNB	59.7%
Linear Model	SGDClassifier	82.3%
	Logistic Regression	83.2%
SVC	SVC	83.3%
	LinearSVC	**84.2%**
	NuSVC	83.4%

From the result above, it is clear that LinearSVC holds the highest accuracy. However, I found an online resource on how we can combine different algorithms in NLTK, and then it will do the voting system whereby each algorithm gets one vote, and the classification (in this case, positive or negative) that has the most vote is the chosen one.[11] To do this, we write our class function *VoteResult* and our *classify* function. The *classify* function then will iterate through each of the models (classifiers) to determine its result (pos/neg) based on the test set. Each model has one vote. Then, after we are done iterating, we will return the result that has the most vote.

```
class VoteResult(ClassifierI):
    def __init__(self, *classifiers):
        self._classifiers = classifiers

    def classify(self,test_set):
        votes = []
        for c in self._classifiers:
            v = c.predict(test_set)[0]
            votes.append(v)
        return mode(votes)


vote_result = VoteResult(SGD,
                         MultinomialNB,
                         BernoulliNB,
                         LogisticRegression,
                         SVC,
                         LinearSVC,
                         NuSVC)
```

Figure 4.6: *VoteResult* class to return the most voted classification in different models

[11] Python Programming, “Combine Classifier Algorithms”, https://pythonprogramming.net/combine-classifier-algorithms-nltk-tutorial/

Then, I run my *predict_sentiment* function with tester.py, and it shows that the accuracy in this voting system of different models is 83.8% (see index for screenshot). This is lower than the highest accuracy that we have seen in using LinearSVC, which is about 84.2%. This is perhaps because MultinomialNB has a really low accuracy. Then, I tried to remove MultinomialNB from the list of models, but then it gave me an error because now the number of models is an even number instead of an odd number, which caused *mode(votes)* to not work appropriately. Because of this, I decided to go with my LinearSVC model in my submission instead of combining the algorithms and voting.

6. Final conclusion

In summary, using three different methods, I found the following result:

Table 2: Summary of result using different methods

Methods	**Model**	**Accuracy**	**Import time**	**Prediction time**
Method 1: Using *textblob*	Naive Bayes	70.3%	0.0008 seconds	2.7 seconds
Method 2: Using *nltk*, using our own extraction function	Logistic Regression	84.6%	9698 seconds (approximately 2.5 hours)	NA
Method 3: Using *nltk*, using in-built extraction function	LinearSVC	84.2%	22.592 seconds	5.4 seconds

The most important factor in choosing the best method to use is the accuracy rate, then the import time. From the table above, Method 1 has the lowest accuracy, so Method 1 is out of the picture. Then, in terms of accuracy, Method 2 clearly wins because it has 84.6% accuracy. However, its import time is extremely large, which is 2.5 hours, which doesn't meet the assignment requirement that should be below 2 hours. Hence, Method 3 wins since accuracy rate is about as high as Method 2 (around 84.2%, only 0.4% difference from Method 2) with relatively quick import time, which is 22.592 seconds and prediction time of 5.4 seconds.

7. References and acknowledgement

I would like to thank Professor Bodin for his help throughout the entire course and on this particular assignment. Then, I also would like to thank the authors of the following resources, documents, and tutorials that I found online. Without Professor Bodin and these online sources, I will not be able to finish this and truly understand what I am doing.

Professor Bruno Bodin's Week 09: Machine Learning and Week 10: Text Mining notebook.

Learning about machine learning classifier:
"6. Learning to Classify Text." Natural Language Toolkit — NLTK 3.6 Documentation. Accessed April 19, 2021. https://www.nltk.org/book/ch06.html.

Learning about building the model using nltk:
Franklin, S. J. "Sentiment Analysis of Movie Reviews in NLTK Python." Medium. Last modified January 1, 2020. https://medium.com/@joel_34096/sentiment-analysis-of-movie-reviews-in-nltk-python-4af4b76a6f3.

Learning about other types of machine learning algorithm:
"Python Programming Tutorials." Python Programming Tutorials. Accessed April 19, 2021. https://pythonprogramming.net/sklearn-scikit-learn-nltk-tutorial/.

Learning about combining different algorithms and voting for one classification:
"Python Programming Tutorials." Python Programming Tutorials. Accessed April 19, 2021. https://pythonprogramming.net/combine-classifier-algorithms-nltk-tutorial/.

Learning about textblob implementation on sentiment analysis:
"Textblob.en.sentiments — TextBlob 0.16.0 Documentation." TextBlob: Simplified Text Processing — TextBlob 0.16.0 Documentation. Accessed April 19, 2021. https://textblob.readthedocs.io/en/dev/_modules/textblob/en/sentiments.html#NaiveBayesAnalyzer.train.

8. Index

8.1 Method 1: Python Code

Python code:
https://drive.google.com/file/d/1x1wXtkymmG8ePigp8hvmP5ObhromkR3m/view?usp=sharing

8.2 Method 2: Python Notebook and Code

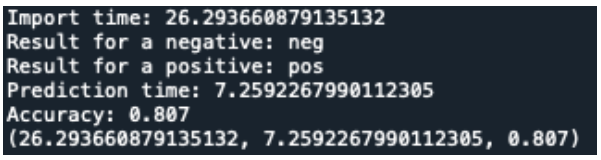
Python code:
https://drive.google.com/file/d/1x1wXtkymmG8ePigp8hvmP5ObhromkR3m/view?usp=sharing

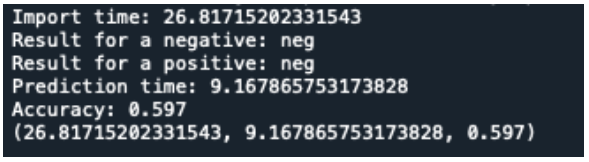
Python notebook:
https://drive.google.com/file/d/1poeO-wjfnZV5bjuw9-5Kz_DV0QTjs5_B/view?usp=sharing

8.3 Method 3: Python Code

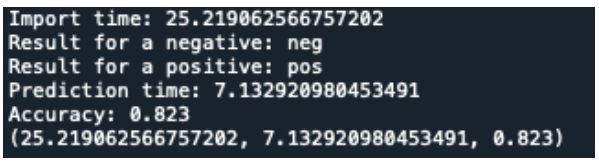
On GitHub classroom submission as sentiment_predict.py

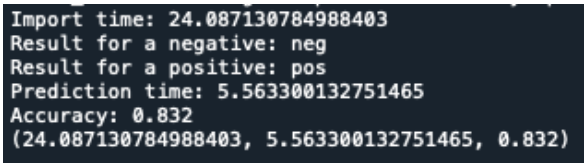
8.4 Method 3: Accuracy results for different models



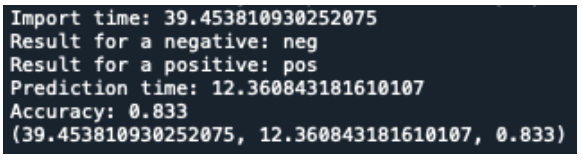


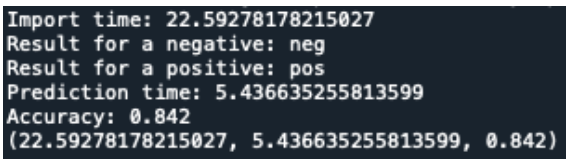
Left: Using MultinomialNB (accuracy: 80.7%)
Right: Using BernoulliNB (accuracy: 59.7%)



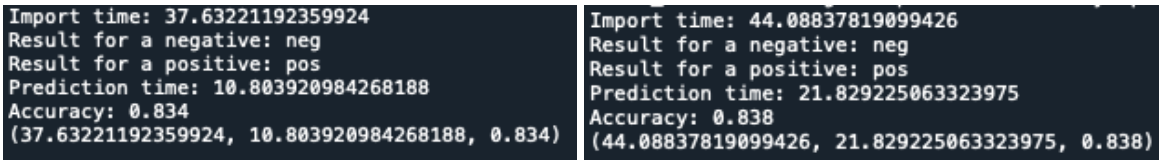


Left: Using SGDClassifier (accuracy: 82.3%)
Right: Using logistic regression (accuracy: 83.2%)





Left: Using SVC (accuracy: 83.3%)
Right: Using Linear SVC (accuracy: 84.2%)



Left: Using NuSVC (accuracy: 83.4%)
Right: Combining algorithms and voting (accuracy: 83.8%)