

CS 588-01 INTRO TO BIG DATA COMPUTING

PROJECT REPORT

**SENTIMENTAL ANALYSIS ON AMAZON REVIEWS DATA USING SUPERVISED
LEARNING**

**Poornashree Ganapathi Subramanian (A25341403), Yukta J Gujjar (A25339938), Bhuvana
Muttanapalli (A25328556)**

University of Alabama, Huntsville

TABLE OF CONTENTS

Chapter	Title	Page Number
1	Introduction	3
2	Discussion of Methods	3
3	Result, Data Visualization and Analysis	6
4	Conclusion	9
5	Reference	9
6	Appendix	10

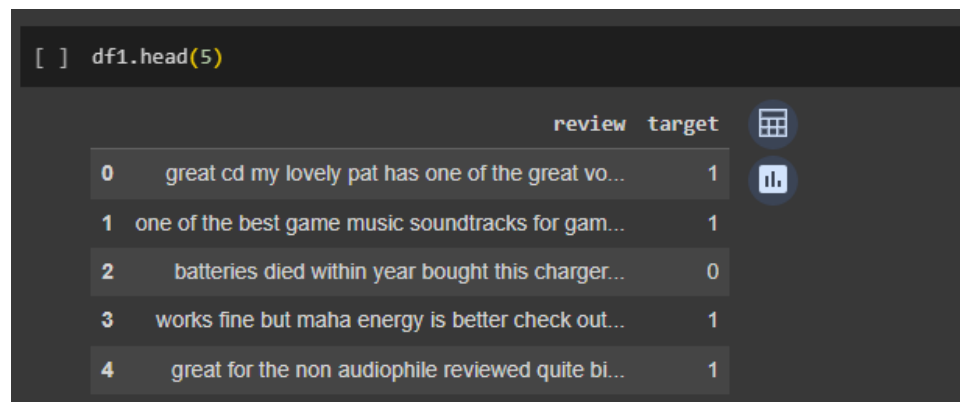
1. Introduction

The use of social media websites has grown significantly during the past few decades. It is used by millions of people to voice their thoughts and ideas on a vast range of subjects. Social networking websites produce a lot of data as a result. Our project uses Amazon reviews as a ready-to-use dataset to analyze sentiment on social media. There are 1.5 million Amazon reviews in the dataset, divided into 500,000 training and 1 million test reviews. The data must be preprocessed to eliminate mistakes and slang words to guarantee excellent data. Preprocessing techniques, such as noise removal and word contraction mapping, will be employed to eliminate special characters, mentions, URLs, and HTML tags. Furthermore, in order to enhance the model's feature representation, contractions found in reviews will be mapped into formal writing. Our project's goal is to enhance sentiment analysis results by reducing errors and guaranteeing a reliable and consistent depiction of user sentiment.

2. Discussion of Methods

2.1 Data Preprocessing

We decided to utilize the ready-to-use social media dataset available on Kaggle. 1.5 million Of the actual Amazon review data is split it into 500 thousand training data and 1 million test data. Now we load the dataset in a .csv format. The data is also converted to DataFrame using pandas library for various tabular operations. The loaded dataset is shown below.



```
[ ] df1.head(5)
```

	review	target
0	great cd my lovely pat has one of the great vo...	1
1	one of the best game music soundtracks for gam...	1
2	batteries died within year bought this charger...	0
3	works fine but maha energy is better check out...	1
4	great for the non audiophile reviewed quite bi...	1

Fig 2.1 Amazon review dataset

All the values in the dataset are not always important. Many dirty/garbage values, noisy values can also be present in the dataset. These values work as outliers which harm our prediction. The preprocessing steps include removal of noisy data, spelling correction, lemmatization and word-stemming. Special characters, mentions, URLs, and HTML tags are all acceptable in reviews, but they are not useful for sentiment analysis. Those have to be eliminated and the contractions must be mapped into formal writing. Input: "I don't realllly like the product! :(". After removing the noisy data, spelling correction and word contraction mapping: "I do not really like the product".

Reducing a word's inflectional forms to a single base or root is the primary goal of lemmatization and word-stemming. These two approaches are not the same, though. The way word-stemming algorithms operate is by removing frequent suffixes and prefixes. Lemmatization, on the other hand, considers the

words' morphological forms. Example: **Stemming:** "qualities" → "qualiti" **Lemmatization:** "qualities" → "quality".

2.2 Feature Extraction

The Amazon review dataset consists of texts of reviews by customers and has been classified as positive or negative. The dataset would be largely balanced with both positive and negative reviews from the Amazon webpage. Naturally, terms like "bad," "suck," and "sad" tend to show up in negative class. Conversely, positive class terms like "love," "cool," and "awesome" are commonly used. It's interesting to note how often the word "love" comes up in both classes. That seems weird, but if we look deeper, we see that the actual negative review with the word "love" also contains the word "miss," or the word was just used sarcastically.



Fig 2.2 Sample Negative Class



Fig 2.3 Sample Positive Class

We chose to experiment with three distinct word representations in order to produce the features for the dataset. An N-gram is a text data representation technique that takes n consecutive sequences from the text data that is provided. The reviews were converted into feature vectors by the Scikit-learn library's TfidfVectorizer function, which will be utilized as input estimators. It is a method of extracting the features from the text data. TF stands for Term Frequency and IDF stands for Inverse Document Frequency. It has a dictionary that gives each word a unique index by converting it to a feature index in the matrix. The TfidfVectorizer sets the value based on the frequency of the word in the corpus and increases it proportionately to the count. Three distinct word representations the uni-gram, bi-gram, and tri-gram were employed. As opposed to using each word in a unigram, the results demonstrate that using word sequences in bi- and tri-gram improves model performance. The result of feature extraction and model performance is given below.

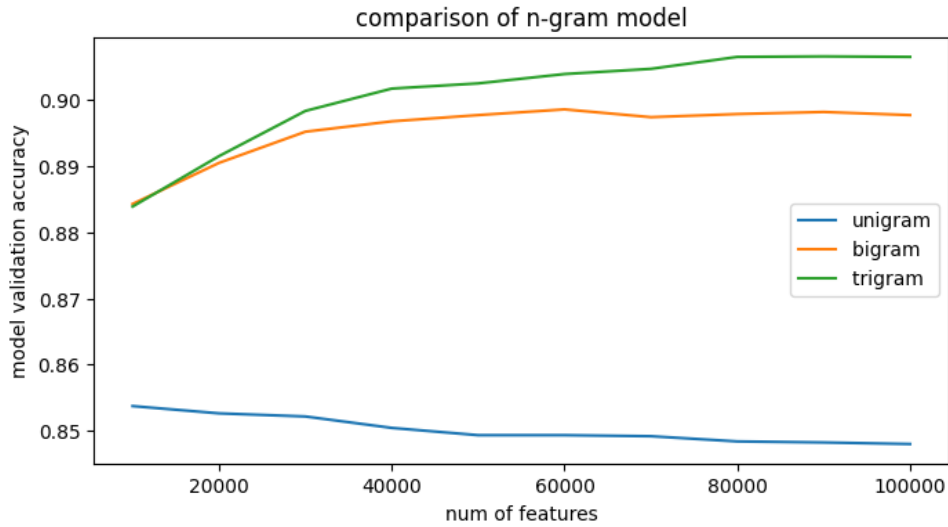


Fig 2.4 N-gram model performance

2.3 Classifiers

We use the pre-processed and extracted data to train our classifier model and evaluate the sentiment analysis's performance. We applied three distinct classifier models: the Convolutional Neural Network, Support Vector Machine, and the Naive Bayes Classifier. Every classifier has specific parameters that must be established, and the parameters' values have an impact on the classifier's overall performance.

Naïve Bayes

Naive Bayes is a powerful classification algorithm rooted in Bayes' theorem, leveraging the assumption of feature independence within classes. Among its variations, Multinomial Naive Bayes stands out for text classification due to its efficiency, speed in processing large datasets, and effectiveness in predicting outcomes. This model, favored for its performance, is readily available in the Scikit-learn library, offering a straightforward implementation for text-based sentiment analysis and classification tasks.

Support Vector Machine

Support Vector Machine (SVM) is a supervised learning method used for classification tasks, establishing a dividing line between data classes by identifying points closest to the line from each class and maximizing the margin between them. When implementing an SVM classifier, selecting the appropriate kernel, like linear, RBF, or polynomial, is crucial as it determines the model's performance. While the RBF kernel is often a reasonable choice due to its ability to handle non-linear relationships between class labels and features, in text categorization where data tends to be linearly separable, linear kernel is implemented. Optimal kernel parameter selection is vital, achieved through Grid Search with Cross Validation to avoid overfitting. We use the Scikit-learn library to create the Support Vector Classifier (SVC).

Convolution neural network

The Convolution Neural Network (CNN), widely recognized for its success in computer vision tasks, is also applied to text classification, displaying effectiveness in solving Natural Language Processing (NLP) problems. In our project using Amazon review data, we utilized a CNN through the Keras library with a Tensorflow backend. Employing tokenization, we represented Reviews as numerical sequences, limiting their length to 20 words. The model, consisting of four hidden layers, underwent training with categorical cross-entropy as the Loss function and multiple optimization algorithms like Adam, Stochastic Gradient Descent with Momentum, Adagrad, and RMSProp. To prevent overfitting, we incorporated Lasso Regression and Dropout techniques, fine-tuning them manually. In our experiments, models trained with 20 epochs displayed the best performance. Consequently, we chose the RMSProp model as our final CNN architecture for the Amazon review data due to its efficiency and efficacy.

3. Result, Data Visualization and Analysis

For the Multinomial Naïve Bayes classifier, we calculated the Receiver Operating Characteristics (ROC) Curve and the Confusion Matrix. A 0.95 area was shown by the ROC curve. According to the confusion matrix, there are over a hundred and seventy thousand true negative records, twenty thousand false positive records, over a hundred and seventy thousand true positive records, and around twenty thousand false positive records. The overall accuracy of this classifier was 89.37%.

Accuracy	89.37%	
Class	Precision	Recall
Negative	0.8945	0.8929
Positive	0.8931	0.8947

Table 1: Naïve Bayes classification result

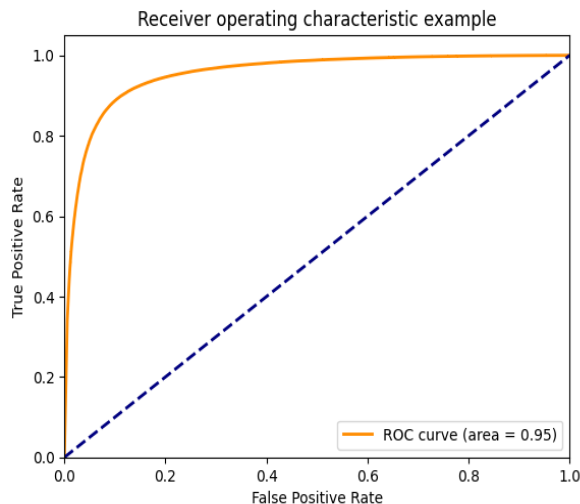


Fig 3.1: ROC curve- Naïve Bayes classifier

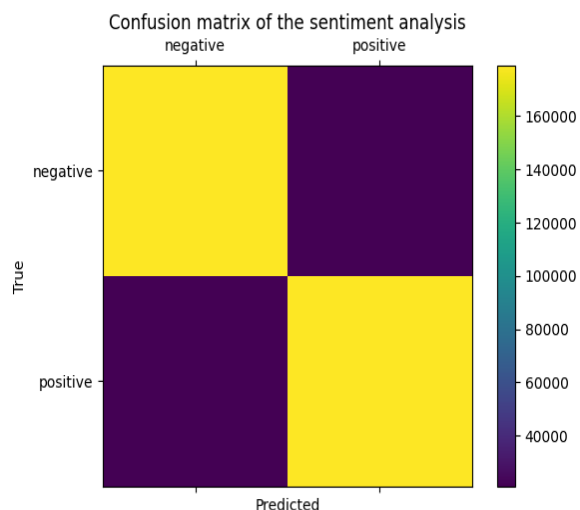


Fig 3.2: Confusion Matrix

We used the SVM classifier model using a linear kernel on this project. The results show that the SVM classifier with a linear kernel generates a better classification than the Naive bayes classifier, linear kernel has a better score on both classification accuracy and ROC score. We calculated the Receiver Operating Characteristics (ROC) Curve and the Confusion Matrix. A 0.98 area was shown by the ROC curve. According to the confusion matrix, there are over a hundred and eighty thousand true negative records, twenty thousand false positive records, over a hundred and eighty thousand true positive records, and around twenty thousand false positive records. The overall accuracy of this classifier was 93.753%.

Accuracy	93.753%	
Class	Precision	Recall
Negative	0.9377	0.9373
Positive	0.9373	0.9378

Table 2: **SVM classification result**

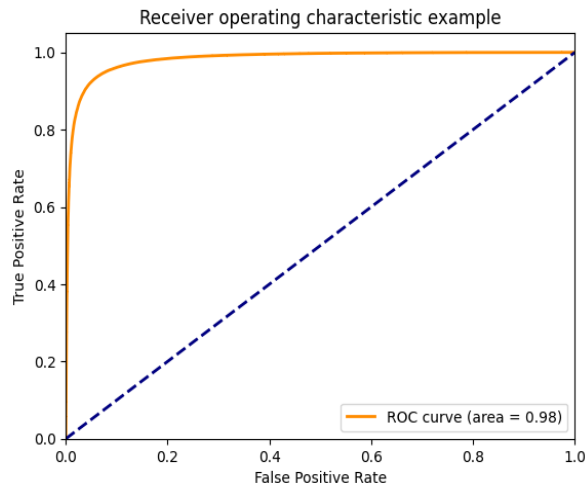


Fig 3.3: ROC curve- SVM **classifier** - Linear Kernel

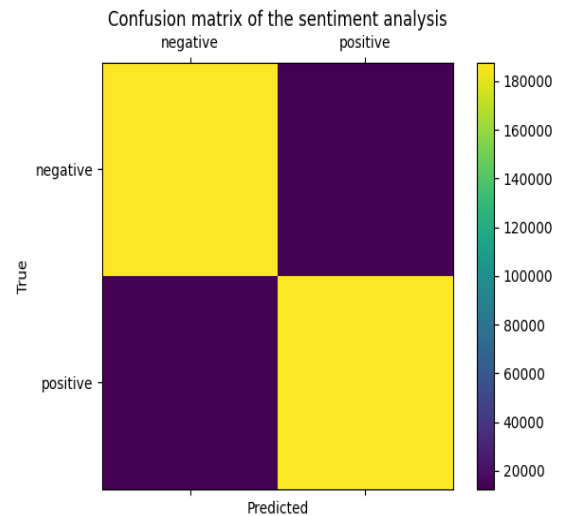


Fig 3.4: Confusion Matrix

We included four hidden convolution layers in our CNN model. Since the data representation is in a 1-dimensional sequence, a 1-dimensional convolution layer was applied. The CNN model produced a well-classified result. In contrast to our initial expectation that the CNN model would perform better, the CNN model performs slightly lower than our SVM model when compared to the performance metrics. We calculated the Receiver Operating Characteristics (ROC) Curve and the Confusion Matrix. A 0.97 area was shown by the ROC curve. According to the confusion matrix, there are over a hundred and seventy thousand true negative records, twenty-five thousand false positive records, over a hundred and seventy thousand true positive records, and around twenty-five thousand false positive records. The overall accuracy of this classifier was 90.09%.

Accuracy	90.09%	
Class	Precision	Recall
Negative	0.9594	0.8372
Positive	0.8556	0.9646

Table 3: **CNN classification result**

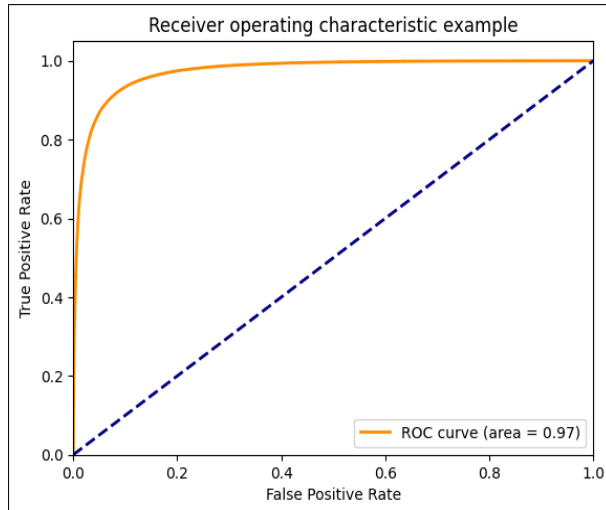


Fig 3.3: ROC curve- CNN classifier

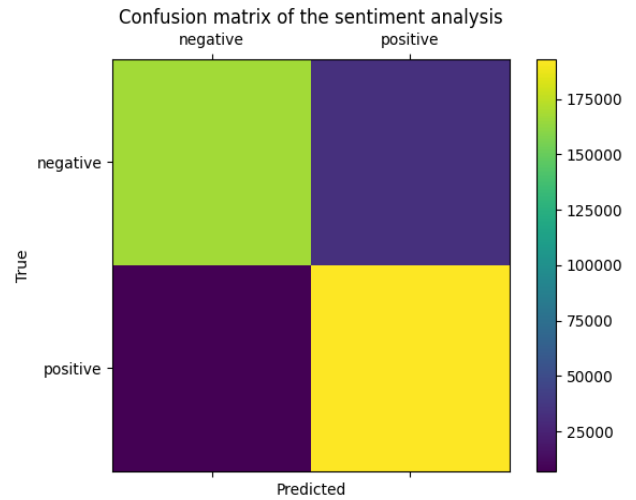


Fig 3.6: Confusion Matrix

Consequently, four CNN models with various optimization functions have been trained. Figure 3.7 shows a plot of the validation accuracy during the training procedure. This result indicates that the convergence of Adam, RMSProp, and Adagrad is faster. According to the experiment's findings, the models that trained using RMSProp 20 epochs are the best. Since the RMSProp model is the most efficient, we decided to use it for our final CNN.

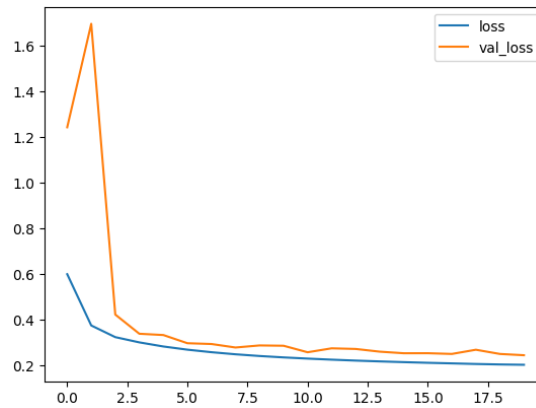


Fig 3.7 Validation Accuracy

4. Conclusion

Due to the nature of Amazon Review dataset which has richer words and longer texts, our models succeeded in performing really well to predict the sentiment. The Naive Bayes classifier managed to achieve accuracy of 89.37%, SVM gave 93.753% and CNN gave 90.09%. Again, linear SVM classifier has the best performance, beating the CNN classifier. This is because the data is linearly separable thus linear model beat even the Neural Network model. From the results of all the classifiers, we found that the best algorithm for the sentiment analysis on Amazon reviews dataset is the SVM with linear kernel. The data collection, data preprocessing and feature extraction process are highly crucial for the model to give it excellent performance since the data is the most important thing on all machine learning-related problems. As for designing and implementing the model, after the preprocessing and feature extraction it is easier to train the Naive Bayes and SVM model. Since they only need a small number of parameters to be tuned. On the other hand, training the Neural Network model is hard as it has so many parameters that need to be trained namely learning rate, regularization rate, padding size and the number of hidden layers.

5. Reference

1. Andreea Salinca. "Convolutional Neural Networks for Sentiment Classification on Business Reviews". In: arXiv (2017).
2. Korovkinas, K., & Danėnas, P. (2017). SVM and Naïve Bayes Classification Ensemble Method for sentiment analysis. *Baltic Journal of Modern Computing*, 5(4).
3. Alhashmi, S. M., Khedr, A. M., Arif, I., & El Bannany, M. (2021). Using a Hybrid-Classification Method to Analyze Twitter Data During Critical Events. *IEEE Access*, 9, 141023–141035.
4. Y. Xu, X. Wu, and Q. Wang. Sentiment analysis of yelps ratings based on text reviews, 2015.
5. Shaikh, T., & Deshpande, D. (2016). Feature Selection Methods in Sentiment Analysis and Sentiment Classification of Amazon Product Reviews. *International Journal of Computer Trends and Technology*, 36(4), 225–230.

6. Appendix

1. Loading the dataset and preprocessing

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from textblob import TextBlob
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_curve, auc
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
from time import time

df_train_data = pd.read_csv('C:\\Users\\FIONA_SHREK\\big-data-project\\data\\train_reviews.csv')
df_test_data = pd.read_csv('C:\\Users\\FIONA_SHREK\\big-data-project\\data\\test_reviews.csv')

df_train_data.columns = ['review', 'target']
df_train_data = df_train_data.sample(frac = 0.177)
df_train_data.shape

(637196, 2)

df_test_data.columns = ['review', 'target']
df_test_data.shape

(399997, 2)

X_train, X_validation, y_train, y_validation = train_test_split(df_train_data.review, df_train_data.target, test_size = 0.02, random_state = 1234)

print("training data has {} entries. {1:.2f}% positive and {2:.2f}% negative".format(len(y_train), len(y_train[y_train==1])/len(y_train)*100, len(y_train[y_train==1])/len(y_train)))
print("validation data has {} entries. {1:.2f}% positive and {2:.2f}% negative".format(len(y_validation), len(y_validation[y_validation==1])/len(y_validation)))

training data has 624452 entries. 50.05% positive and 49.95% negative
validation data has 12744 entries. 50.09% positive and 49.91% negative
```

2. Stemming and Lemmatization

```
[32]: from nltk.stem import WordNetLemmatizer

[33]: normalizer = WordNetLemmatizer()

[34]: import nltk
      nltk.download('wordnet')

[nltk_data] Downloading package wordnet to
[nltk_data]   C:\\Users\\FIONA_SHREK\\AppData\\Roaming\\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!

[34]: True

[36]: df_train_data_2 = df_train_data.copy()
      df_train_data_2['review'] = [' '.join([normalizer.lemmatize(word) for word in t.split()]) for t in df_train_data_2['review'].tolist()]
      X_train, X_validation, y_train, y_validation = train_test_split(df_train_data.review, df_train_data.target, test_size = 0.02, random_state = 1234)

[ ]: res_trigram_lemma = compare_clf_nfeature(X_train, X_validation, y_train, y_validation, ngram_range=(1,3))

[37]: from nltk.stem import PorterStemmer

[38]: normalizer = PorterStemmer()

[39]: df_train_data_2 = df_train_data.copy()
      df_train_data_2['review'] = [' '.join([normalizer.stem(word) for word in t.split()]) for t in df_train_data_2['review'].tolist()]
      X_train, X_validation, y_train, y_validation = train_test_split(df_train_data.review, df_train_data.target, test_size = 0.02, random_state = 1234)

[40]: df_train_data_2.tail()
```

	review	target
920682	great life stori about how not to live thi boo...	1
2161629	review of faith of our father thi book doe not...	0
112280	gave up tri to play after hour of frustrat it ...	0
3237337	worth everi penni am veri happi with thi made ...	1
1949451	terrif remix as an avid fan the song are remix...	1

3.Feature Extraction

```
[37]: from sklearn.naive_bayes import MultinomialNB
      from sklearn.linear_model import LogisticRegression

[38]: vectorizer = CountVectorizer()
      clf = MultinomialNB()
      n_features = np.arange(10000, 100001, 10000)

[39]: def calc_accuracy(pipeline, X_train, X_test, y_train, y_test):
      t0 = time()
      sentiment_clf = pipeline.fit(X_train, y_train)
      y_pred = sentiment_clf.predict(X_test)
      train_test_time = time() - t0
      accuracy = accuracy_score(y_test, y_pred)
      fpr = dict()
      tpr = dict()
      roc_auc = dict()

      fpr, tpr, _ = roc_curve(y_test, y_pred, pos_label=1)
      roc_auc = auc(fpr, tpr)

      print("model accuracy : {0:.2f}".format(accuracy))
      print("model auc : {0:.2f}".format(roc_auc))
      print("train validation time : {0:.2f}s".format(train_test_time))
      print("=====")

      return accuracy, roc_auc, train_test_time

[40]: def compare_clf_nfeature(X_train, X_test, y_train, y_test, vectorizer=vectorizer, clf=clf, n_features=n_features, stop_words=None, ngram_range = (1,1)):
      res = []
      print(clf)
      for feature in n_features:
          vectorizer.set_params(stop_words=stop_words, ngram_range=ngram_range, max_features=feature)
          acc_pipeline = Pipeline([('vectorizer', vectorizer), ('classifier', clf)])
          print("result on model with {} features".format(feature))
          accuracy, auc, time = calc_accuracy(acc_pipeline, X_train, X_test, y_train, y_test)
          res.append((feature, accuracy, auc, time))

      return res

def compare_clf_nfeature(X_train, X_test, y_train, y_test, vectorizer=vectorizer, clf=clf, n_features=n_features, stop_words=None, ngram_range = (1,1)):
    res = []
    print(clf)
    for feature in n_features:
        vectorizer.set_params(stop_words=stop_words, ngram_range=ngram_range, max_features=feature)
        acc_pipeline = Pipeline([('vectorizer', vectorizer), ('classifier', clf)])
        print("result on model with {} features".format(feature))
        accuracy, auc, time = calc_accuracy(acc_pipeline, X_train, X_test, y_train, y_test)
        res.append((feature, accuracy, auc, time))

    return res

df_word_freq = pd.read_csv('C:\\Users\\FIONA_SHREK\\big-data-project-master\\data\\word_freq.csv')
df_word_freq.columns = ['word', 'neg', 'pos', 'total']
df_word_freq.head(25)
```

```
[30]:
```

	word	neg	pos	total
0	to	313164	252567	565731
1	the	257870	266013	523883
2	not	238226	103119	341345
3	my	190845	125979	316824
4	it	157482	147804	305286
5	and	153972	149649	303621
6	you	103890	198340	302230
7	is	135021	111667	246688
8	in	115543	101163	216706
9	for	99003	117372	216375
10	of	92739	91098	183837
11	on	84194	84197	168391
12	that	82860	83157	166017
13	me	91862	71600	163462
14	have	90105	66293	156398
15	so	88518	65630	154148
16	do	80973	53944	134917
17	but	84920	48606	133526
18	just	64016	62953	126969
19	with	50153	65180	115333
20	be	58616	54480	113096
21	at	62732	49625	112357
22	can	64738	46719	111457
23	was	63955	47463	111418
24	this	53000	41574	94574

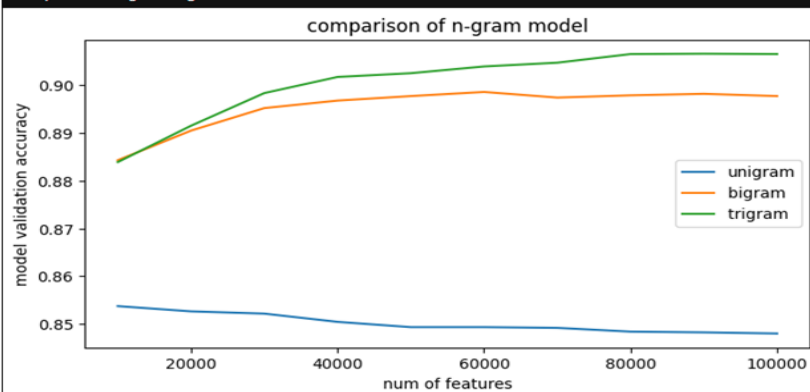
```
[42]: X_train, X_validation, y_train, y_validation = train_test_split(df_train_data.review, df_train_data.target, test_size = 0.02, random_state = 1234)
tf_vectorizer = TfidfVectorizer()
res_tfidf_trigram = compare_clf_nfeature(X_train, X_validation, y_train, y_validation, ngram_range=(1,3), vectorizer=tf_vectorizer)

MultinomialNB()
result on model with 10000 features
model accuracy : 0.88
model auc : 0.88
train validation time : 410.85s
=====
result on model with 20000 features
model accuracy : 0.89
model auc : 0.89
train validation time : 383.57s
=====
result on model with 30000 features
model accuracy : 0.90
model auc : 0.90
train validation time : 374.96s
=====
result on model with 40000 features
model accuracy : 0.90
model auc : 0.90
train validation time : 389.17s
=====
result on model with 50000 features
model accuracy : 0.90
model auc : 0.90
train validation time : 393.01s
=====
result on model with 60000 features
model accuracy : 0.90
model auc : 0.90
train validation time : 386.99s
=====
result on model with 70000 features
model accuracy : 0.90
model auc : 0.90
train validation time : 394.52s
=====
result on model with 80000 features
model accuracy : 0.91
model auc : 0.91
train validation time : 442.46s
```

```
cols = ['n_features', 'val_acc', 'val_auc', 'time']
res_tfidf_unigram = pd.DataFrame(res_tfidf_unigram, columns=cols)
res_tfidf_bigram = pd.DataFrame(res_tfidf_bigram, columns=cols)
res_tfidf_trigram = pd.DataFrame(res_tfidf_trigram, columns=cols)
```

```
cols = ['n_features', 'val_acc', 'val_auc', 'time']
res_tfidf_unigram = pd.DataFrame(res_tfidf_unigram, columns=cols)
res_tfidf_bigram = pd.DataFrame(res_tfidf_bigram, columns=cols)
res_tfidf_trigram = pd.DataFrame(res_tfidf_trigram, columns=cols)
plt.figure(figsize=(8,4))
plt.plot(res_tfidf_unigram.n_features, res_tfidf_unigram.val_acc, label='unigram')
plt.plot(res_tfidf_bigram.n_features, res_tfidf_bigram.val_acc, label='bigram')
plt.plot(res_tfidf_trigram.n_features, res_tfidf_trigram.val_acc, label='trigram')
plt.title("comparison of n-gram model")
plt.xlabel("num of features")
plt.ylabel("model validation accuracy")
plt.legend()
```

<matplotlib.legend.Legend at 0x1e3bb8d3b80>



```
[33]: len(y_train[y_train==1])/len(y_train)
```

```
[33]: 0.5005108479114487
```

```
[34]: tb_sentiment = [TextBlob(i).sentiment.polarity for i in X_validation]
tb_sentiment = [0 if x < 0 else 1 for x in tb_sentiment]
```

```
[35]: cm = confusion_matrix(y_validation, tb_sentiment)
cm
```

```
[35]: array([[2653, 3708],
        [ 240, 6143]], dtype=int64)
```

```
[36]: print("accuracy = {:.2f}".format(accuracy_score(y_validation, tb_sentiment)))
print(classification_report(y_validation, tb_sentiment))
```

```
accuracy = 0.69
          precision    recall  f1-score   support

     0       0.92      0.42      0.57      6361
     1       0.62      0.96      0.76      6383

   accuracy          0.69          12744
  macro avg       0.77      0.69      0.67      12744
 weighted avg       0.77      0.69      0.67      12744
```

4. Naïve Bayes model

```
[ ]: #Naive Bayes classifier

[26]: def plot_cm(cm):
        labels = ['negative', 'positive']
        fig = plt.figure()
        ax = fig.add_subplot(111)
        cax = ax.matshow(cm)
        plt.title('Confusion matrix of the sentiment analysis')
        fig.colorbar(cax)
        ax.set_xticklabels([''] + labels)
        ax.set_yticklabels([''] + labels)
        plt.xlabel('Predicted')
        plt.ylabel('True')
        plt.show()

[43]: X_train, y_train = df_train_data.review, df_train_data.target
        X_test, y_test = df_test_data.review, df_test_data.target

[44]: clf = MultinomialNB()
        vectorizer = CountVectorizer(ngram_range=(1,3), max_features=90000)
        X_train = vectorizer.fit_transform(X_train)
        X_test = vectorizer.transform(X_test)

[45]: clf.fit(X_train, y_train)
        probas = clf.predict_proba(X_test)
        y_pred = np.argmax(probas, axis=-1)
        y_pred_prob = [i[1] for i in probas]
        acc_score = accuracy_score(y_test, y_pred)
        cm = confusion_matrix(y_test, y_pred)

        print(acc_score)
        print(cm)

0.893779203344025
[[178580  21420]
 [ 21068 178929]]

[46]: print(classification_report(y_test, y_pred, digits=4))
        fpr = dict()
        tpr = dict()
        roc_auc = dict()

        fpr, tpr, _ = roc_curve(y_test, y_pred_prob, pos_label=1)
        roc_auc = auc(fpr, tpr)

        plt.figure()
        lw = 2
        plt.plot(fpr, tpr, color='darkorange',
                  lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
        plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic example')
        plt.legend(loc="lower right")
        plt.show()

        plot_cm(cm)
```

	precision	recall	f1-score	support
0	0.8945	0.8929	0.8937	200000
1	0.8931	0.8947	0.8939	199997
accuracy			0.8938	399997
macro avg	0.8938	0.8938	0.8938	399997
weighted avg	0.8938	0.8938	0.8938	399997

5.SVM with linear kernel

```
# SVM Classifier with linear kernel
```

```
def plot_auc(label, prediction):
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    fpr, tpr, _ = roc_curve(label, prediction, pos_label=1)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

```
from sklearn.calibration import CalibratedClassifierCV

def sentiment_svm(train, test, C, frac = 1, max_features = None, kernel='linear', ngram_range=(1,3), gamma=0, use_params=False):
    X_train, y_train = train.sample(frac=frac, random_state=1234).review, train.sample(frac=frac, random_state=1234).target
    X_test, y_test = test.sample(frac=frac, random_state=1234).review, test.sample(frac=frac, random_state=1234).target

    tf_vectorizer = TfidfVectorizer(ngram_range=ngram_range, max_features=max_features)
    X_train = tf_vectorizer.fit_transform(X_train)
    X_test = tf_vectorizer.transform(X_test)

    if kernel == 'linear':
        if use_params:
            params = find_params(X_train, y_train)
            print ('Best params: %s' % (params))
            svm = CalibratedClassifierCV(base_estimator=LinearSVC(C=params['C']), cv=5)
        else:
            svm = CalibratedClassifierCV(base_estimator=LinearSVC(C=C), cv=5)
    print(svm)

    t0 = time()
    svm.fit(X_train, y_train)
    train_test_time = time() - t0
    print("training time: {0:.2f}s".format(train_test_time))

    probas = svm.predict_proba(X_test)
    y_pred = np.argmax(probas, axis=-1)
    y_pred_prob = [i[1] for i in probas]

    acc_score = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    print("Model accuracy = {0:.3f}%".format(acc_score*100))
    print(classification_report(y_test, y_pred, digits=4))

    plot_auc(y_test, y_pred_prob)
    plot_cm(cm)
```



```

sentiment_svm(df_train_data, df_test_data, 0.5)
CalibratedClassifierCV(base_estimator=LinearSVC(C=0.5), cv=5)
C:\Users\FIONA_SHREK\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\calibration.py:300: FutureWarning: 'base_estimator' was renamed to
'estimator' in version 1.2 and will be removed in 1.4.
  warnings.warn(
C:\Users\FIONA_SHREK\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of 'dual' will
change from 'True' to 'auto' in 1.5. Set the value of 'dual' explicitly to suppress the warning.
  warnings.warn(
C:\Users\FIONA_SHREK\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of 'dual' will
change from 'True' to 'auto' in 1.5. Set the value of 'dual' explicitly to suppress the warning.
  warnings.warn(
C:\Users\FIONA_SHREK\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of 'dual' will
change from 'True' to 'auto' in 1.5. Set the value of 'dual' explicitly to suppress the warning.
  warnings.warn(
C:\Users\FIONA_SHREK\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of 'dual' will
change from 'True' to 'auto' in 1.5. Set the value of 'dual' explicitly to suppress the warning.
  warnings.warn(
C:\Users\FIONA_SHREK\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of 'dual' will
change from 'True' to 'auto' in 1.5. Set the value of 'dual' explicitly to suppress the warning.
  warnings.warn(
training time: 214.54s
Model accuracy = 93.753%

```

	precision	recall	f1-score	support
0	0.9377	0.9373	0.9375	200000
1	0.9373	0.9378	0.9375	199997
accuracy			0.9375	399997
macro avg	0.9375	0.9375	0.9375	399997
weighted avg	0.9375	0.9375	0.9375	399997

6. Neural Network -CNN

```

#CNN

from keras.layers import *
from keras.models import Model
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.utils import shuffle

X_train = list(df_train_data.review)
X_test = list(df_test_data.review)

y_train = [[1,0] if x == 0 else [0,1] for x in df_train_data.target]
y_test = [[1,0] if x == 0 else [0,1] for x in df_test_data.target]

X_train, y_train = shuffle(X_train, y_train)
X_test, y_test = shuffle(X_test, y_test)

y_train = np.array(y_train)
y_test = np.array(y_test)

max_feature = 5000
maxlen = 100
embed_size = 25

tokenizer = Tokenizer(num_words=max_feature)
tokenizer.fit_on_texts(X_train)

token_train = tokenizer.texts_to_sequences(X_train)
token_test = tokenizer.texts_to_sequences(X_test)

X_train_final = pad_sequences(token_train, maxlen=maxlen, padding='post')
X_test_final = pad_sequences(token_test, maxlen=maxlen, padding='post')

```



```

from keras import optimizers
from keras import regularizers
eta = 1
maxlen=100
input = Input(shape=(maxlen,))
net = Embedding(max_feature, embed_size)(input)
net = Dropout(0.2)(net)
net = BatchNormalization()(net)

net = Conv1D(16, 8, padding='same', activation='relu')(net)
net = Dropout(0.2)(net)
net = BatchNormalization()(net)
net = Conv1D(16, 4, padding='same', activation='relu')(net)
net = Dropout(0.2)(net)
net = BatchNormalization()(net)
net = Conv1D(16, 4, padding='same', activation='relu')(net)
net = Dropout(0.2)(net)
net = BatchNormalization()(net)
net = Conv1D(16, 4, padding='same', activation='relu')(net)
net = Dropout(0.2)(net)
net1 = BatchNormalization()(net)

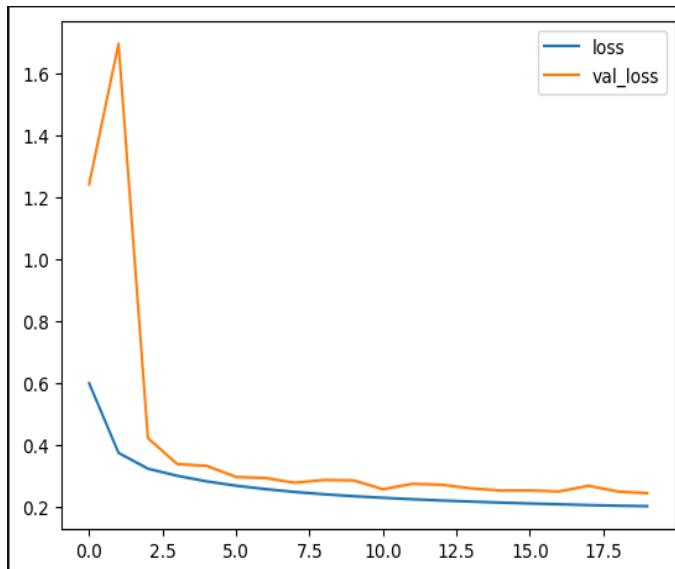
net = Conv1D(2, 1)(net)
net = GlobalAveragePooling1D()(net)
output = Activation('softmax')(net)
model = Model(inputs = input, outputs = output)
ada = optimizers.legacy.Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
model.compile(optimizer=ada, loss='categorical_crossentropy', metrics=['acc'])
model.summary()

```

```

train_res = model.fit(X_train_final, y_train, batch_size=2048, epochs=20, validation_split=0.1, callbacks=[plot_losses])
acc = train_res.history['val_acc']
print('Accuracy: {}'.format(np.mean(acc)))

```



```

281/281 [=====] - 225s 801ms/step - loss: 0.2037 - acc: 0.9203 - val_loss: 0.2456 - val_acc: 0.9026
Accuracy: 0.8451122075319291

```

```

y_prob = model.predict(X_test_final)
y_pred = y_prob.argmax(axis=-1)
y_test_raw = [0 if (x == [1,0]) else 1 for x in y_test]
y_pred_prob = [i[1] for i in y_prob]
plot_auc(y_test_raw, y_pred_prob)
plot_cm(confusion_matrix(y_test_raw, y_pred))
print(classification_report(y_test_raw, y_pred, digits=4))

```

