Sentiment Analysis of Movie Reviews

Group project

Presented by :

Melanya Khachatryan

Anzhel Torosyan

Narek Mikayelyan

Introduction

- Sentiment analysis, a subset of Natural Language Processing (NLP), aims to extract insights from textual data. It finds applications in various industries, including entertainment and human resources. Our project focuses on sentiment analysis of movie reviews.

Project Goal

- Our primary objective is to determine whether a movie review reflects a positive or negative sentiment.
- We employ machine learning algorithms, particularly the Naive Bayes classifier, for this classification task.

Dataset selection

Consists of 50,000 reviews from IMDb.

Balanced distribution of 25,000 positive and 25,000 negative reviews.

Split into training and testing sets.
Training set contains 80% of the data.
Testing set contains 20% of the data.

Split facilitates model evaluation.

Data processing steps

Removal of Noise:

- Eliminated mentions, URLs, hashtags, and emojis to strip away unnecessary information.
- Removed HTML tags to streamline the dataset.

Standardization:

- Converted the text to lowercase for uniformity.
- Transformed emojis into text equivalents to retain valuable sentiment information.

Data Refinement:

- Eliminated punctuation and stop words to focus on meaningful content.
- Implemented text tokenization to break down sentences into smaller, manageable tokens, typically individual words.

	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The filming tec...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1
...	...	...
49995	I thought this movie did a down right good job...	1
49996	Bad plot, bad dialogue, bad acting, idiotic di...	0
49997	I am a Catholic taught in parochial elementary...	0
49998	I'm going to have to disagree with the previou...	0
49999	No one expects the Star Trek movies to be high...	0

	review	sentiment
0	one of the other reviewers has mentioned that ...	1
1	a wonderful little production the filming tech...	1
2	i thought this was a wonderful way to spend ti...	1
3	basically theres a family where a little boy j...	0
4	petter matteis love in the time of money is a ...	1
5	probably my alltime favorite movie a story of ...	1
6	i sure would like to see a resurrection of a u...	1
7	this show was an amazing fresh innovative ide...	0
8	encouraged by the positive comments about this...	0
9	if you like original gut wrenching laughter yo...	1

	review	sentiment
0	one reviewers mentioned watching 1 oz episode ...	1
1	wonderful little production filming technique ...	1
2	thought wonderful way spend time hot summer we...	1
3	basically theres family little boy jake thinks...	0
4	petter matteis love time money visually stunni...	1
5	probably alltime favorite movie story selfless...	1
6	sure would like see resurrection dated seahunt...	1
7	show amazing fresh innovative idea 70s first a...	0
8	encouraged positive comments film looking forw...	0
9	like original gut wrenching laughter like movi...	1

Bag of Words (BoW) Feature Extraction

- **Feature extraction** is the process of transforming raw input data into a format that can be used for analysis.
- In our case we have selected **Bag of Words (BoW)** technique due to it’s more simplicity compared with **Word2Vec** which usually produces better results
- **Bag of Words (BoW)** works by representing text data as a multiset of words, disregarding grammar and word order but retaining information about word frequency.
- **BoW** operates on the principle of tokenization, wherein each document's text is broken down into individual words or tokens. These tokens are then represented as a sparse matrix, where rows correspond to documents and columns represent unique words found in the corpus.

Bag of Words (BoW) Feature Extraction

In general we can divide the **Bag of Words (BoW)** feature extraction process into two steps.

Step 1. It involves fitting the CountVectorizer to the training data using the fit_transform() method.
This creates a sparse matrix representing each document's word occurrences.

In Step 2. The same vectorizer is used to transform the test data, ensuring consistency in vocabulary across datasets.

Bag of words is used for vectorization

```
1  cv = CountVectorizer(min_df=300)
2  X_train_bow=cv.fit_transform(X_train['review'])
3  X_test_bow=cv.transform(X_test['review'])
4  print("Shape of X_train_bow:", X_train_bow.shape)
5  print("Shape of X_test_bow:", X_test_bow.shape)
6  X_train_bow
```

```
Shape of X_train_bow: (40000, 2091)
Shape of X_test_bow: (10000, 2091)

<40000x2091 sparse matrix of type '<class 'numpy.int64'>'
        with 2590912 stored elements in Compressed Sparse Row format>
```

Naïve Bayes Classification

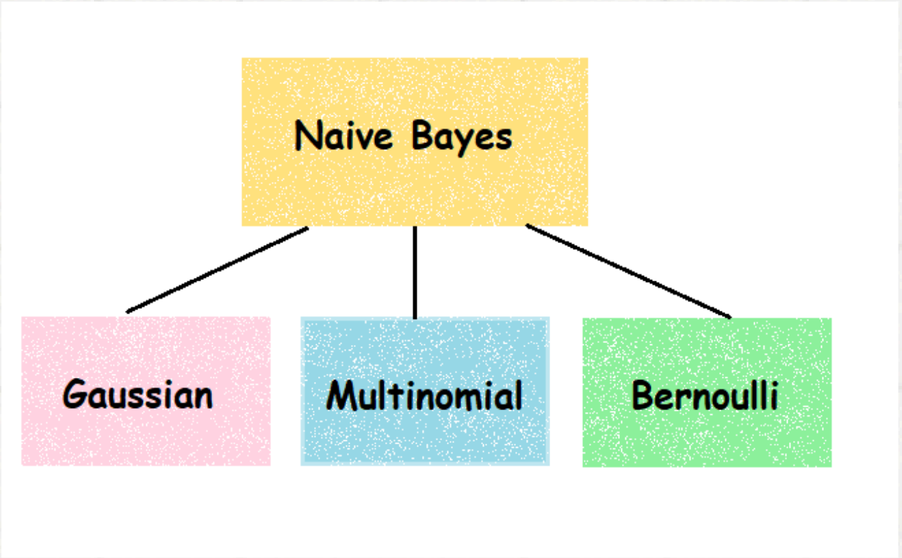
PRIOR

$$P(y = c) = log[\ (\sum_{x \ni T} xi + \alpha) / [\ (\sum_{i=1}^{n} xi + 2\alpha)]$$

Class Conditional

$$P(X|label) = log[\ (\sum_{x \ni T} xi + \alpha) / [\ (\sum_{i=1}^{n} xi + n\alpha)]$$

- $\sum_{x \ni T} xi$ – **# of times word i** appears in a sample of label y
- $\sum_{i=1}^{n} xi$ – total **# of all the words** belonging to label y
- α – Laplace smoothing parameter
- n – size of the dictionary (the unique words) (n = 2106)



Class label for a new point from testing set:

$$\hat{y}MAP = argmax\ log\ P(y \mid x)$$

```
        self.prior = {}
        self.class_conditional = defaultdict(lambda: defaultdict(int))
        self.param = param

    def train(self, X_train_bow, y_train):
        total_samples = len(y_train)

        # Calculate prior probabilities
        for label in set(y_train):
            self.prior[label] = (np.sum(y_train == label) + self.param) / (total_samples + 2 * self.param)

        # Calculate class conditional probabilities
        for i in range(X_train_bow.shape[0]):
            for word_index in range(X_train_bow.shape[-1]):
                self.class_conditional[word_index][y_train.iloc[i]] += X_train_bow[i, word_index]

    def posterior(self, X_test_bow):
        y_pred = []
        for x in X_test_bow:
            # Initializing posterior with prior probabilities
            probabilities = {label: np.log(self.prior[label]) for label in self.prior}
            for word_index in x.indices:
                for label in self.prior:
                    probabilities[label] += np.log((self.class_conditional[word_index][label] + self.param) /
                                                   (sum(self.class_conditional[word_index].values()) +
                                                    len(self.class_conditional) * self.param))
            y_pred.append(max(probabilities, key=probabilities.get))
        return y_pred
```

Comparison with Sk-learn based built in MultiNomial NaiveBayes Classification

The tables below show the metrics for the equally chosen parameter (alpha =2) for both implementations. However, even with different parameters (1 for our model, 2 for sk-learn based) the difference was intangible (< 1%).

Metrics for Our NB_Classifer

	Precision	Recall	F1 score
0	0.81	0.89	0.84
1	0.88	0.79	0.83
Accuracy score			83.82%

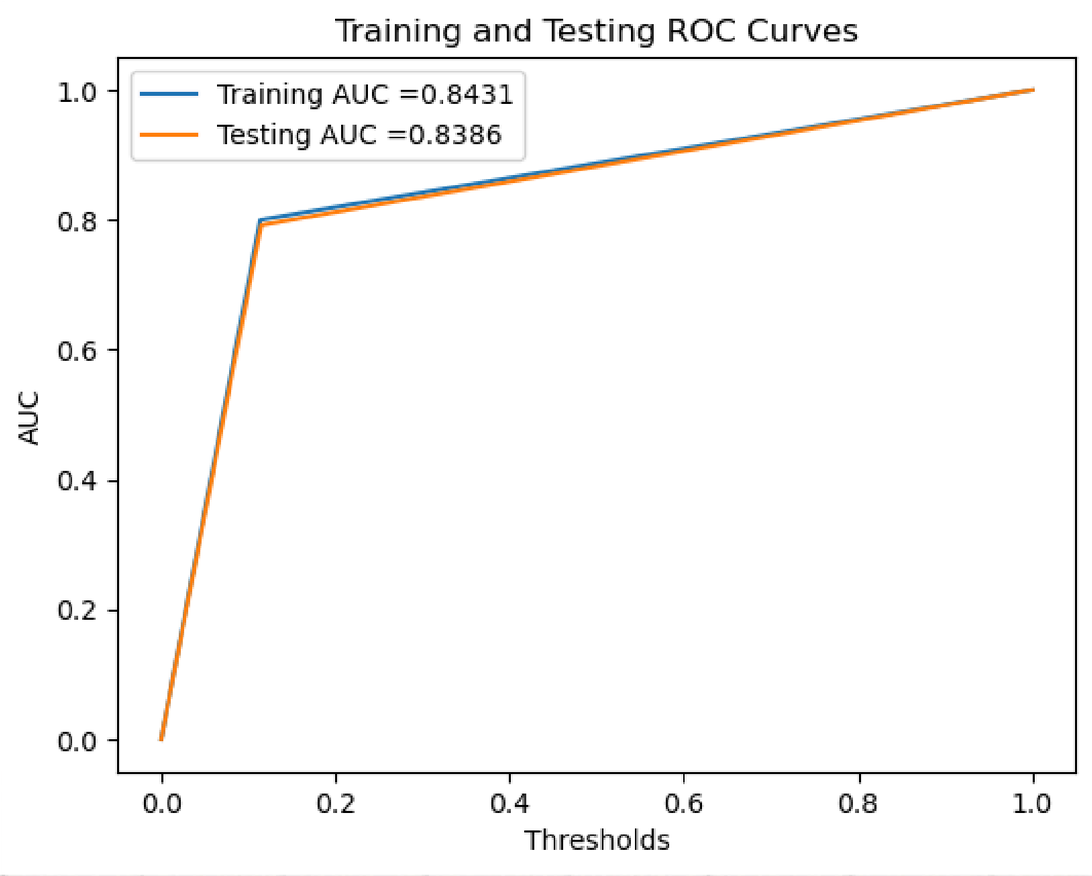
Metrics for Our SK-learn NB Classifier

	Precision	Recall	F1 score
0	0.84	0.84	0.84
1	0.84	0.84	0.84
Accuracy score			84.19%

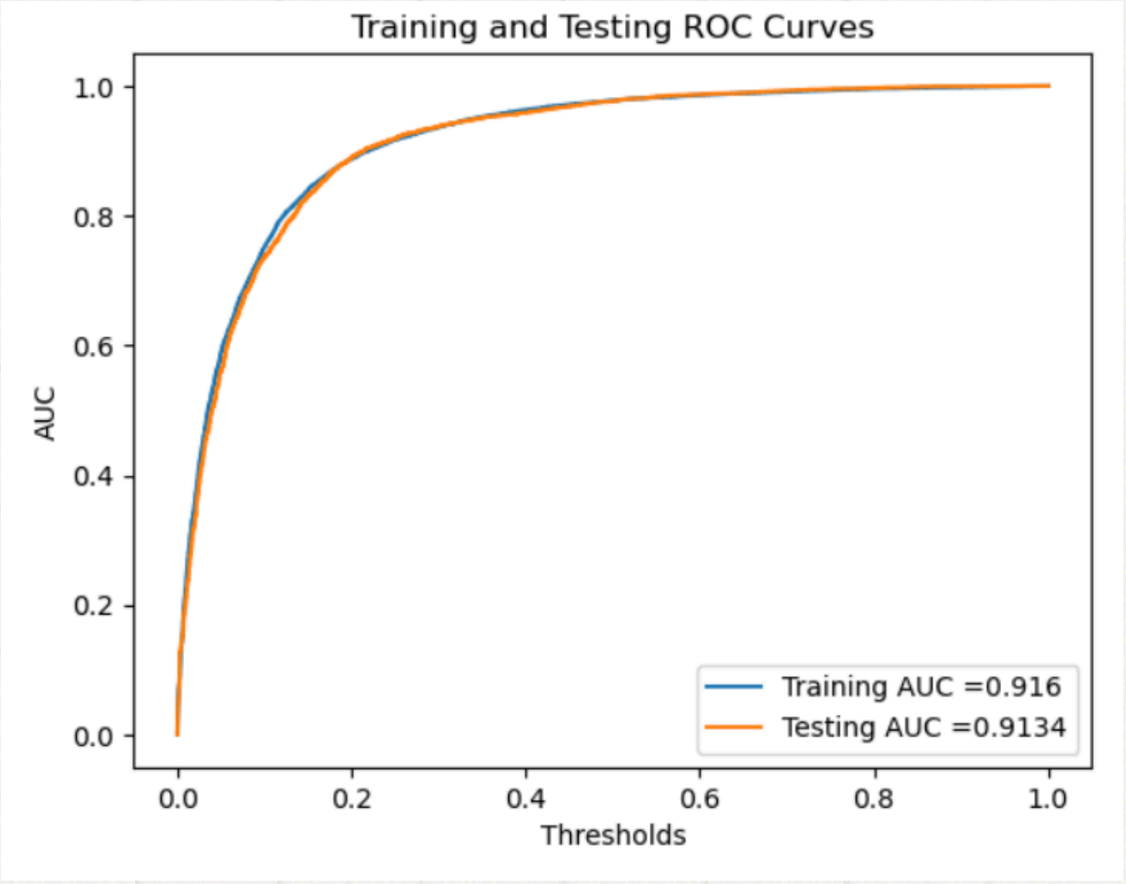
```
1, 0, 1, 0, 1, 0
1, 1, 0, 1, 0, 1
0, 1, 1, 1, 1, 1
1, 0, 0, 1, 1, 0
1, 0, 1, 1, 1, 1
0, 0, 0, 1, 1, 1
1, 0, 1, 1, 0, 0
0, 0, 0, 1, 0, 0
0, 1, 0, 1, 0, 0
0, 0, 0, 0, 0, 0
1, 1, 1, 1, 1, 1
1, 0, 0, 1, 0, 0
1, 1, 0, 1, 0, 0
1, 0, 1, 0, 1, 0
0, 1, 1, 1, 1, 0
1, 0, 0, 1, 0, 0
```

ROC curves for NB with BOW and with sklearn

ROC curves for NB with BOW

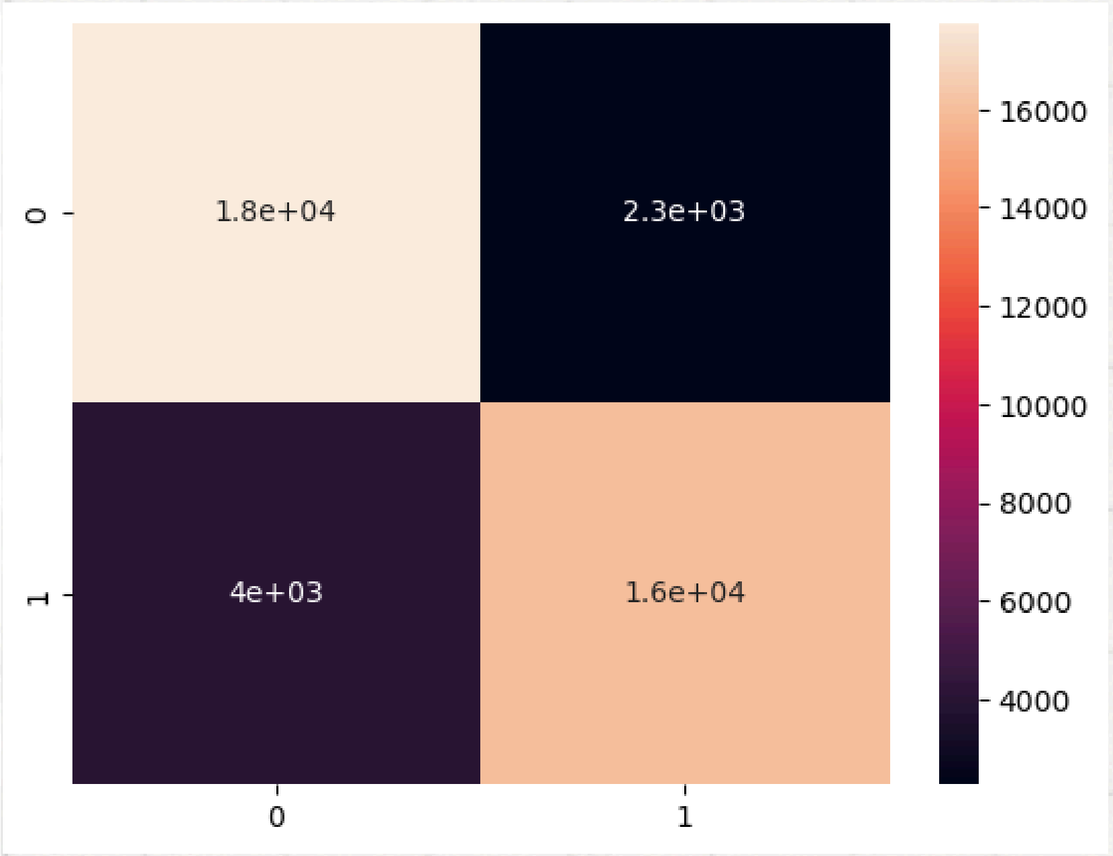


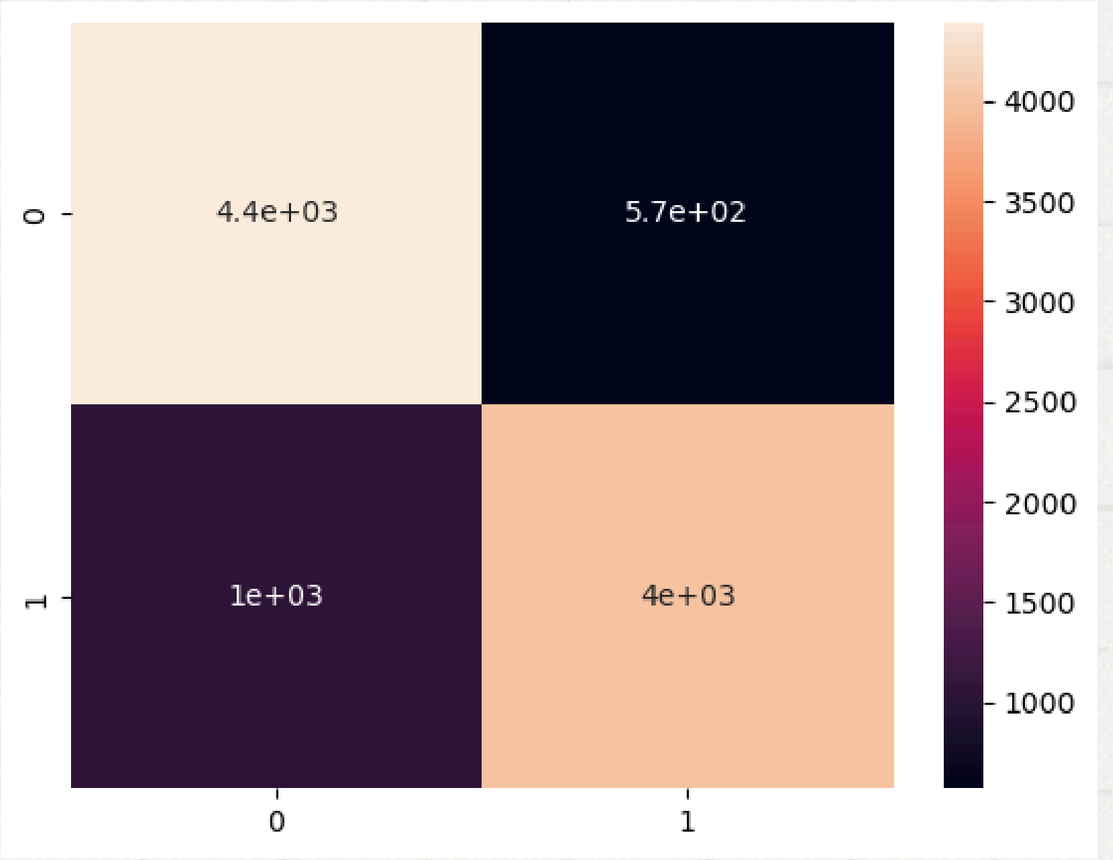
ROC curves for sklearn



Confusion Matrices visualization

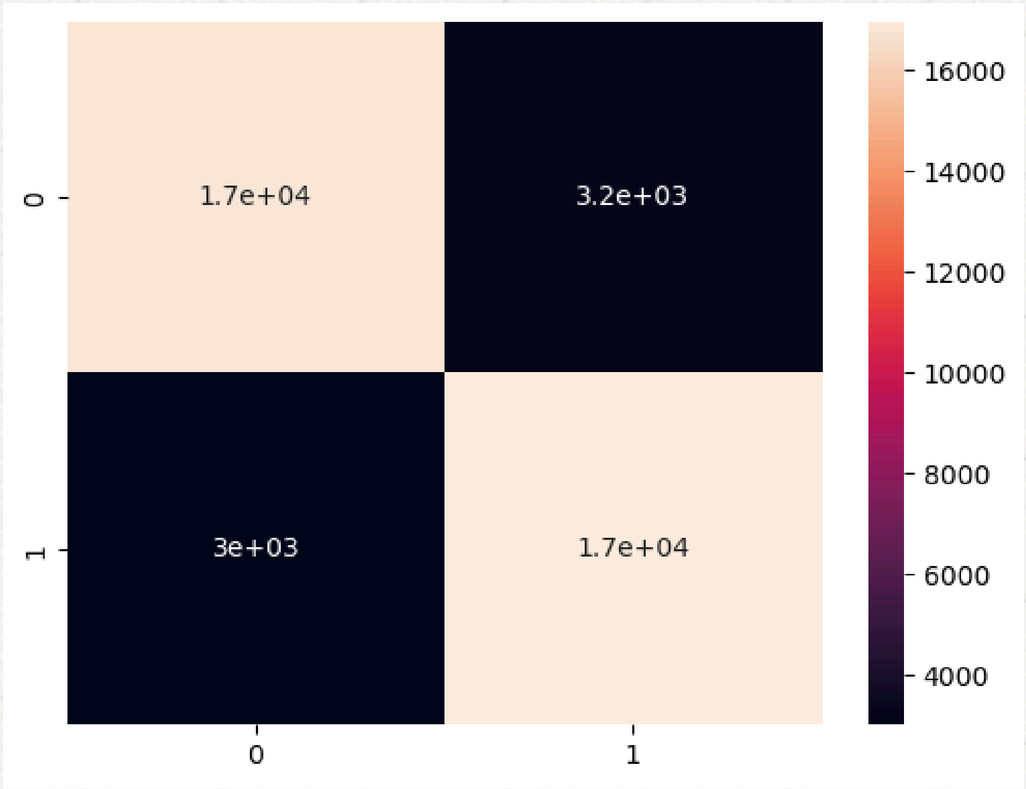
Confusion matrices for NB with BOW

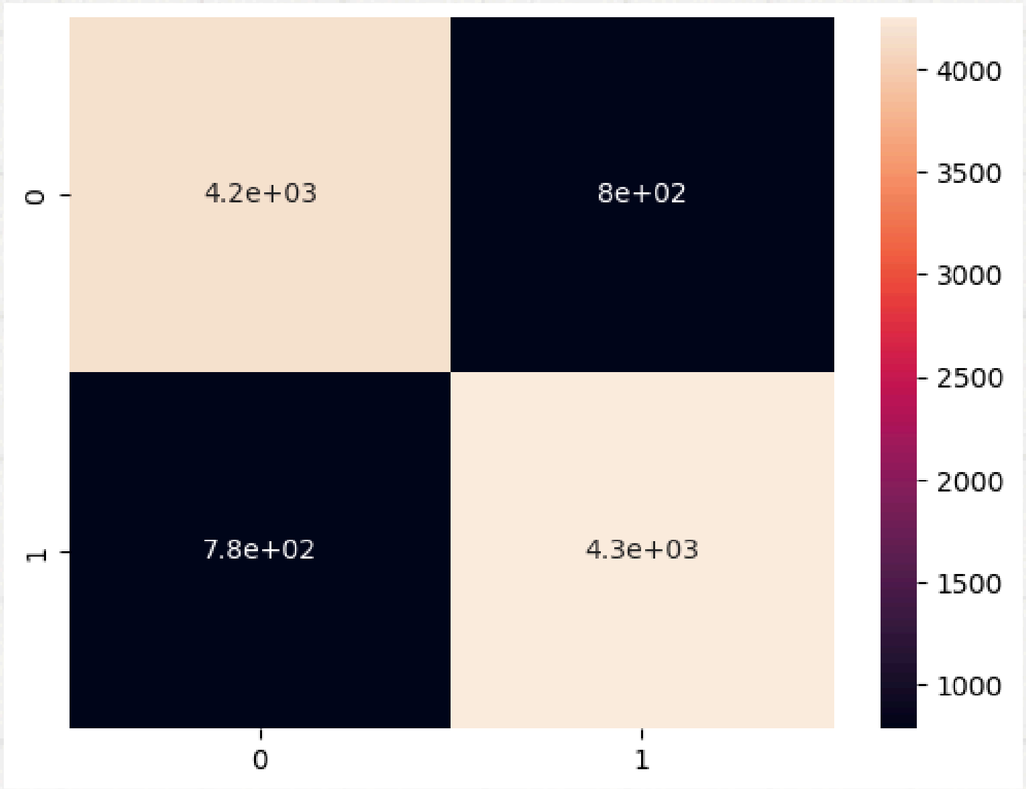




Confusion Matrices visualization

Confusion matrices for sklearn





Thank you!