Sentiment Analysis on IMDb Movie Reviews

Foundations of Machine Learning

Simone Murari
VR502597
University of Verona

Contents

Motivation and Rationale

This machine learning project, titled "Sentiment Analysis on IMDb Movie Reviews," falls within the established research themes of Natural Language Processing (NLP), particularly sentiment analysis and opinion mining. It addresses the well-documented problem of extracting subjective information (opinions and sentiments) from large volumes of textual data, in this case movie reviews on the IMDb platform, and, using a classifier, accurately predicting the sentiment of other unseen textual data, in this case other movie reviews.

In my opinion the significance of this project stems from its potential impact on the subject of **understanding public opinion**. In fact, by analyzing the sentiment expressed in movie reviews, the project can offer valuable insights into public perception of various aspects of films, ranging from plot and acting to social themes and cultural representation.
This information holds significant value for:

- Studios and filmmakers: gauging audience reception, informing creative decisions in future productions, and identifying effective marketing strategies.
- Researchers: studying audience preferences, analyzing cultural trends, and investigating the impact of media on society.
- Individual consumers: making informed decisions about movie choices based on the broader sentiment expressed by other viewers.

State of the Art

Looking at the leaderboard here it's evident that the current state of the art for sentiment analysis tasks are **transformers** and in some cases LSTMs. In fact XLNet [1], an unsupervised language representation learning method that employs Transformer-XL as the backbone model, sits at the top with an accuracy of 96.21 and in the top 10 we can find a lot of models based on BERT (Bidirectional Encoder Representations from Transformers) [2], all with an accuracy greater than 95. Other models with 90+ accuracy use neural networks, in particular LSTMs and RNNs.

To tackle this task, however, I chose to steer away from the SOTA models and I opted for these traditional machine learning models:

- Naive Bayes:
- Logistic Regression:
- SVM

These simpler models are less computationally expensive to run than deep learning ones and require a less time expensive training phase while, at the same time, giving good results on sentiment analysis taks.

Objectives

The general objective of the project is to develop a machine learning model for sentiment analysis on IMDb movie reiviews, aiming to classify user sentiments as positive or negative. In order to do that it is necessary to address smaller "sub-problems" that lead to smaller and specific objectives. For example it is of fundamental importance to preprocess the dataset in order to remove "noise" in the dataset generated by unnecessary or wrong words that do not identify the reviews and that are not needed to achieve the general objective.

Methodology

To develop this project I used the Python programming language, in particular version 3.12 but it works on all Python 3.6+ versions. I chose this language because it is probably the most used language to solve NLP problems and has a large community of people that built tools and libraries that helped me a lot. The different libraries I used are:

- **pandas**: *a fast, powerful, flexible and easy to use open source data analysis and manipulation tool.*[1]
  I used it to store and handle the dataset.
- **scikit-learn**: *an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection, model evaluation, and many other utilities.*[2]
  It is a very powerful library and the one I used the most in my project, for example I used it to handle the training-test split of the dataset and it provided the classifiers I used to solve the problem.
- **Jupyter notebook**: *a notebook authoring application, under the Project Jupyter umbrella. Built on the power of the computational notebook format, Jupyter Notebook offers fast, interactive new ways to prototype and explain your code, explore and visualize your data, and share your ideas with others.*[3]
  I used this library instead of creating a specific *.py* file, it really helped me to develop the project faster and I think it makes the code more readable.
- **NLTK**: *a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.*[4]
  This library was very important during the preprocessing phase and the feature extraction phase because it offers multiple tools that help in fixing and extracting features from the dataset.
- **NumPy**: *is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.*[5]
  This library helped me to handle in a fast way the preprocessed dataset and to get qualitative results from the different classifiers.
- **matplotlib**: *Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.*[6]
  I used this library to plot the quantitative results and the Bag-of-Words sample.

In my implemented pipeline, I started with storing the dataset inside a pandas DataFrame (50000, 2), then I divided it into features X and the target variable y where I mapped the sentiment *"negative"* to the number 0 and the sentiment *"postiive"* to the number 1, so the classifiers could handle them without problems. After that I split the features into a training set (X_train (40000, 2)) and a testing set (X_test (10000, 2)) with a 80/20 split. Then the **preprocessing phase** started where I used regex to handle any corrupt reviews in both the training and test set: for example I found that in a lot of reviews there were HTML tags (in particular linebreaks *
*) that didn't add any value to the reviews and would misguide the classifiers into thinking they were important features in the dataset.
After that I went onto the **feature extraction phase** where I extracted the features from the "raw"

[1] https://pandas.pydata.org/
[2] https://scikit-learn.org/stable/getting_started.html
[3] https://jupyter-notebook.readthedocs.io/en/latest/notebook.html
[4] https://www.nltk.org/
[5] https://numpy.org/doc/stable/
[6] https://matplotlib.org/

reviews using the **Bag-of-Words** technique. To implement this technique I used the library scikit-learn, in particular its CountVectorizer class, and the library NLTK. Using the *fit_transform* method of the CountVectorizer the reviews were tokenized, so each review was transformed into a list of words, and then the *stopwords*, words that have a syntactic value but no semantic value, like *the, are, is, which, where...* were removed. The remaining lists of words got turned into a document-term matrix where each review is transformed into a matrix that associates to all the words that appear in all the reviews the number of times they appear in each specific review. This matrix is obviously very large (50000, ~100000) and most of it would be empty, so I decided, after a few tests, to keep only the 5000 most common words in all the reviews and so obtained a (50000, 5000) BoW matrix between training and test set ((40000, 5000) and (10000, 5000)). Finally, in the **classification phase** I trained the models on the training set and then tested them on the test set. These are the models (from scikit-learn):

- MultinomialNB:

  `MultinomialNB` implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.

  The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

  $$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

  where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{n} N_{yi}$ is the total count of all features for class y.

  The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing.

- LogisticRegression:

  For notational ease, we assume that the target y_i takes values in the set $\{0, 1\}$ for data point i. Once fitted, the `predict_proba` method of `LogisticRegression` predicts the probability of the positive class $P(y_i = 1|X_i)$ as

  $$\hat{p}(X_i) = \operatorname{expit}(X_i w + w_0) = \frac{1}{1 + \exp(-X_i w - w_0)}.$$

  As an optimization problem, binary class logistic regression with regularization term $r(w)$ minimizes the following cost function:

  (1)

  $$\min_w C \sum_{i=1}^{n} s_i \left(-y_i \log(\hat{p}(X_i)) - (1 - y_i) \log(1 - \hat{p}(X_i))\right) + r(w),$$

  where s_i corresponds to the weights assigned by the user to a specific training sample (the vector s is formed by element-wise multiplication of the class weights and sample weights).

- SGDClassifier with loss='log_loss':

  Given a set of training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \mathbf{R}^m$ and $y_i \in \mathcal{R}$ ($y_i \in -1, 1$ for classification), our goal is to learn a linear scoring function $f(x) = w^T x + b$ with model parameters $w \in \mathbf{R}^m$ and intercept $b \in \mathbf{R}$. In order to make predictions for binary classification, we simply look at the sign of $f(x)$. To find the model parameters, we minimize the regularized training error given by

  $$E(w, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$$

  where L is a loss function that measures model (mis)fit and R is a regularization term (aka penalty) that penalizes model complexity; $\alpha > 0$ is a non-negative hyperparameter that controls the regularization strength.

  Log Loss: equivalent to Logistic Regression. $L(y_i, f(x_i)) = \log(1 + \exp(-y_i f(x_i)))$.

LinearSVC:

Given training vectors $x_i \in \mathbb{R}^p$, i=1,..., n, in two classes, and a vector $y \in \{1, -1\}^n$, our goal is to find $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$ such that the prediction given by $\text{sign}(w^T \phi(x) + b)$ is correct for most samples.

SVC solves the following primal problem:

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \max(0, 1 - y_i(w^T \phi(x_i) + b)),$$

SGDClassifier with loss='hinge':

Given a set of training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \mathbf{R}^m$ and $y_i \in \mathcal{R}$ ($y_i \in -1, 1$ for classification), our goal is to learn a linear scoring function $f(x) = w^T x + b$ with model parameters $w \in \mathbf{R}^m$ and intercept $b \in \mathbf{R}$. In order to make predictions for binary classification, we simply look at the sign of $f(x)$. To find the model parameters, we minimize the regularized training error given by

$$E(w, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$$

where L is a loss function that measures model (mis)fit and R is a regularization term (aka penalty) that penalizes model complexity; $\alpha > 0$ is a non-negative hyperparameter that controls the regularization strength.

Hinge (soft-margin): equivalent to Support Vector Classification. $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))$.

You can click on the name of each classifier to see more informations about it on scikit-learn.org

Experiments & Results

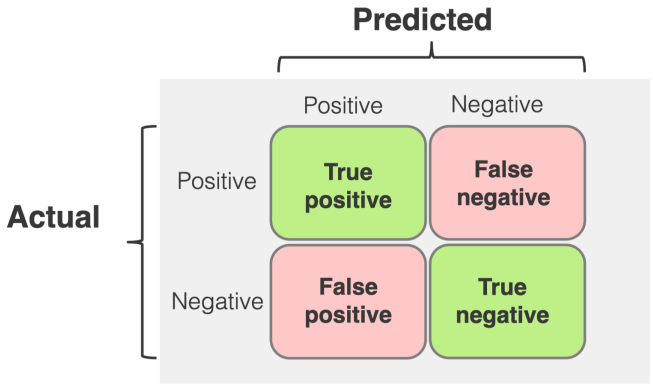
I measured the performance of each classifier first in a qualitative way looking directly at reviews that they predicted correctly and at reviews that they predicted wrongly. This helped me learn that all classifiers struggled the most with very long and ambiguous reviews that were not flat out negative or flat out positive and that analyzed deeply the movie and made use of words that a classifier would expect in both positive and negative reviews. I also looked at the performance of each classifier in a quantitative way, using the library scikit-learn to print the **classification report** and the **confusion matrix**. The classification report provides a text report that shows the main classification metrics: accuracy, precision, recall and f1-score.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

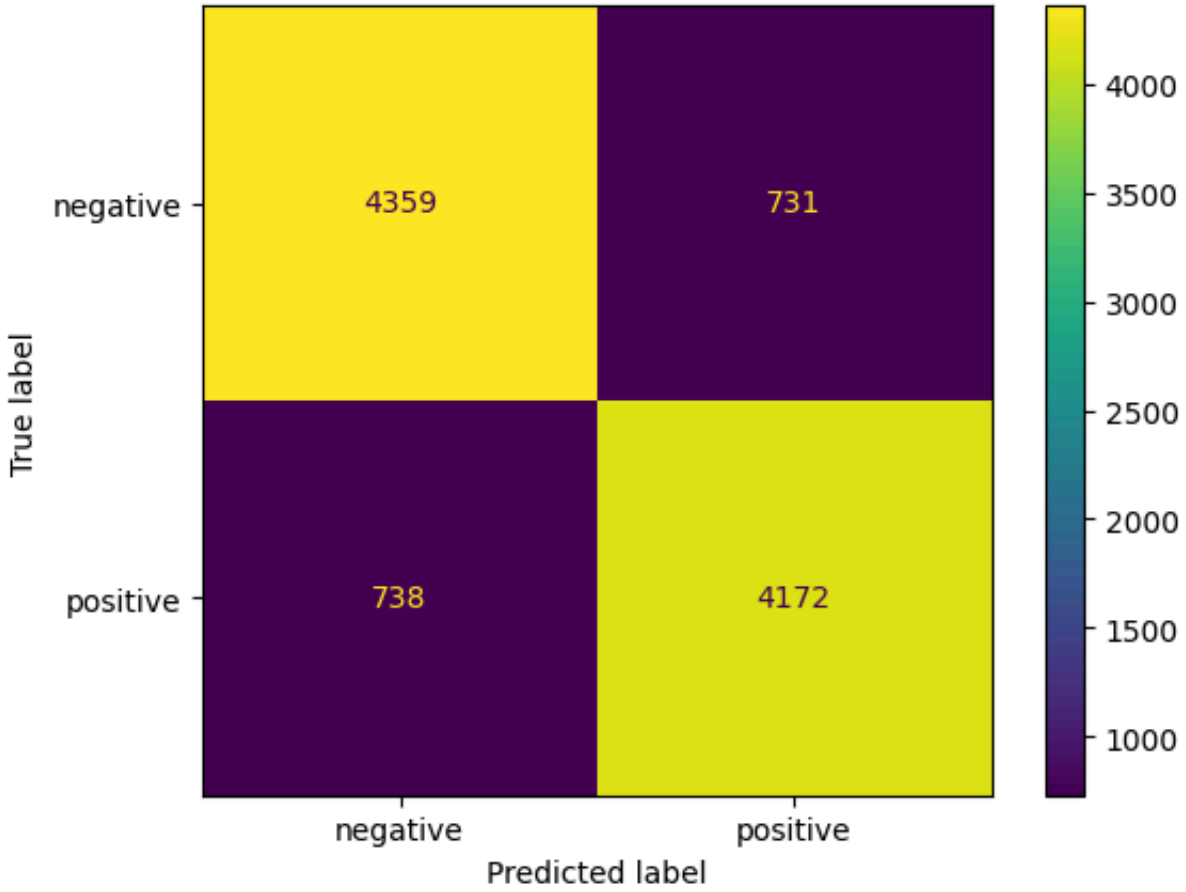
The confusion matrix is a very useful tool that helps in understanding the performance of a classifier. Each row of the matrix represents the true labels while each column represents the predicted labels.



In the next pages you can find the classification report and the confusion matrix for each classifier so you can evalute their performance and see how well they did in the sentiment analysis task on the IMDb Movie Reviews Dataset.

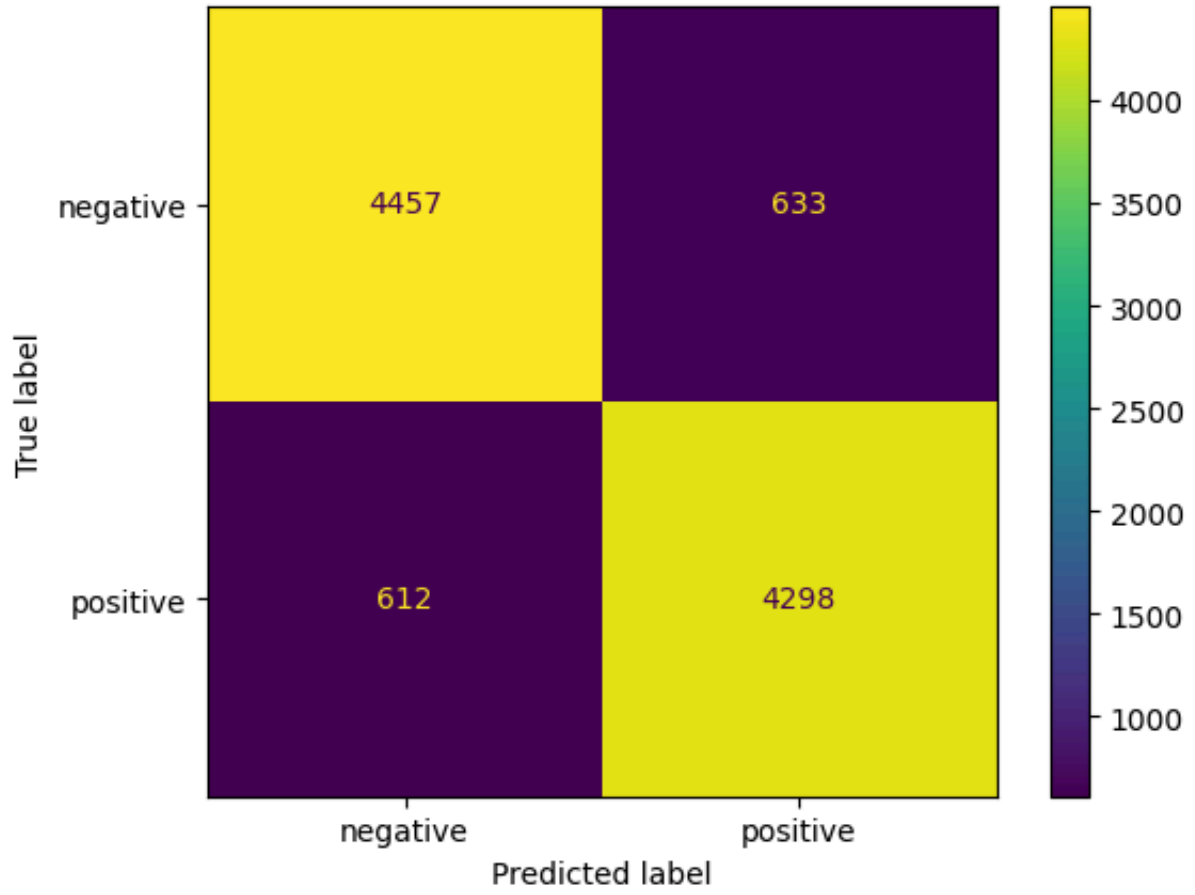
Classification report and confusion matrix for MultinomialNB:

	precision	recall	f1-score	support
negative	0.86	0.86	0.86	5090
positive	0.85	0.85	0.85	4910
accuracy			0.85	10000
macro avg	0.85	0.85	0.85	10000
weighted avg	0.85	0.85	0.85	10000



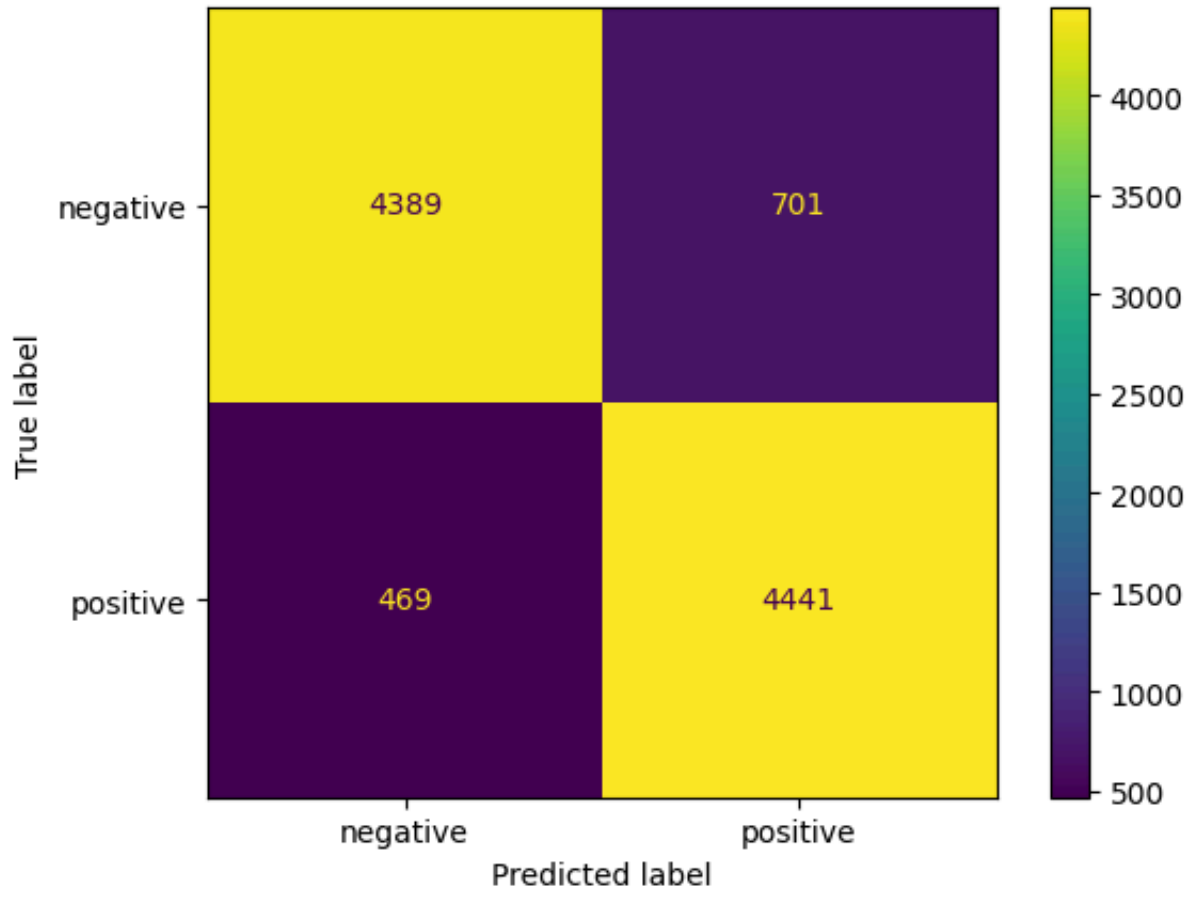
Classification report and confusion matrix for LogisticRegression:

	precision	recall	f1-score	support
negative	0.88	0.88	0.88	5090
positive	0.87	0.88	0.87	4910
accuracy			0.88	10000
macro avg	0.88	0.88	0.88	10000
weighted avg	0.88	0.88	0.88	10000



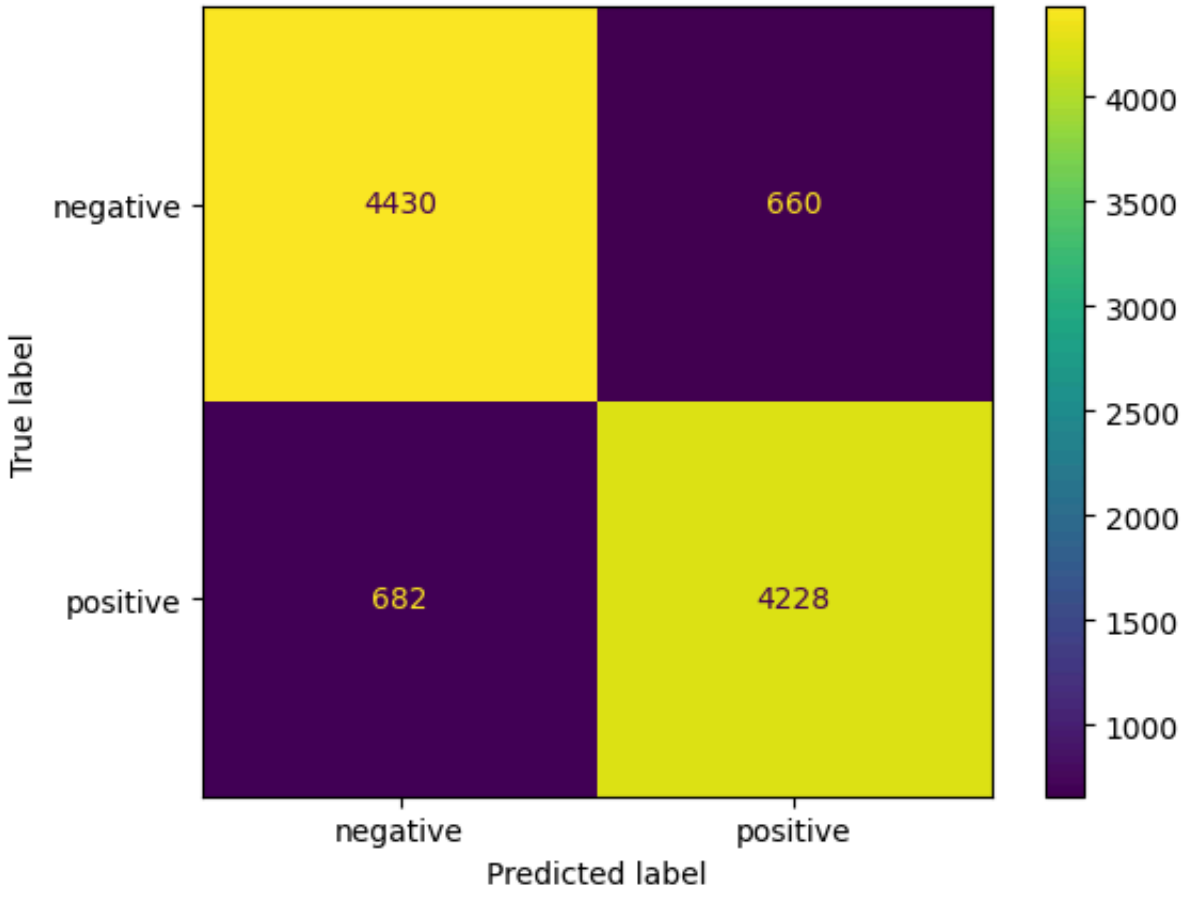
Classification report and confusion matrix for SGDClassifier loss='log_loss':

	precision	recall	f1-score	support
negative	0.90	0.86	0.88	5090
positive	0.86	0.90	0.88	4910
accuracy			0.88	10000
macro avg	0.88	0.88	0.88	10000
weighted avg	0.88	0.88	0.88	10000



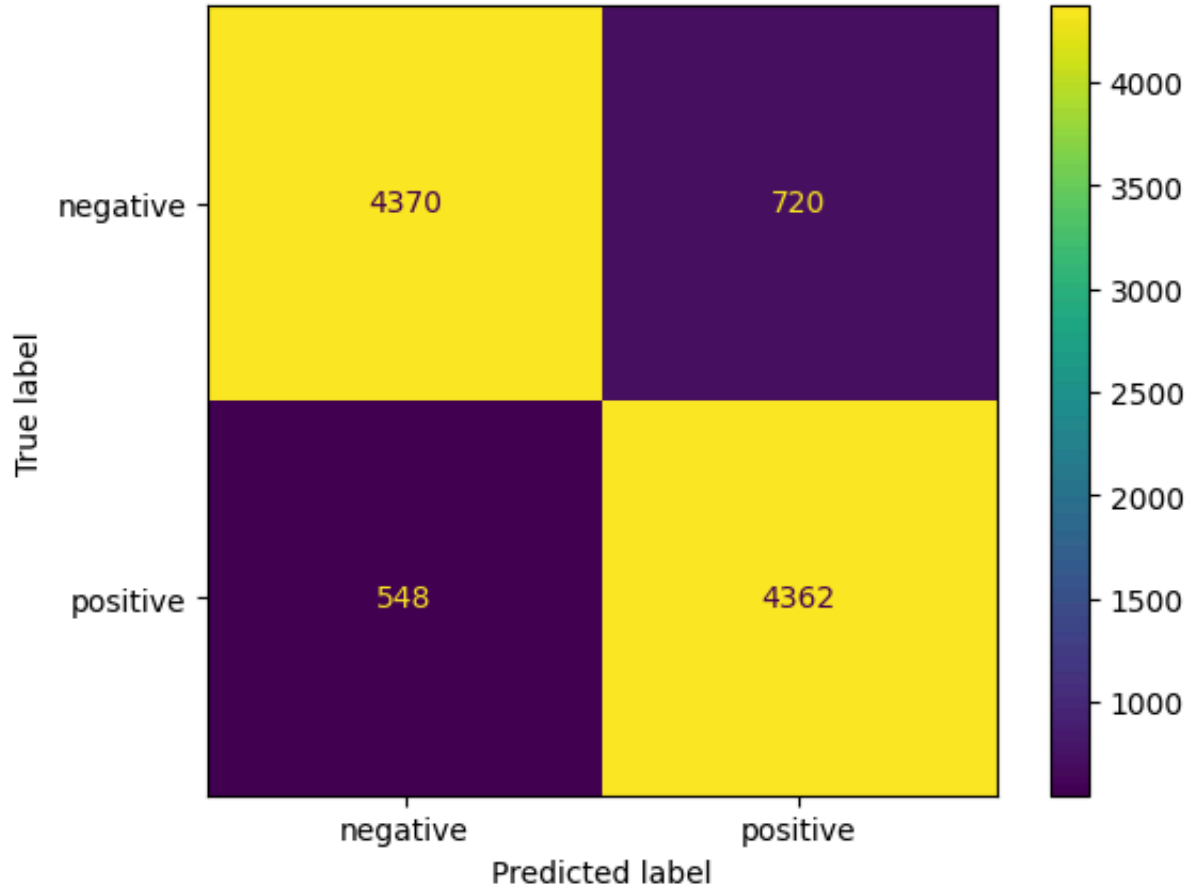
Classification report and confusion matrix for LinearSVC:

	precision	recall	f1-score	support
negative	0.87	0.87	0.87	5090
positive	0.86	0.86	0.86	4910
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000



Classification report and confusion matrix for SGDClassifier loss='hinge':

	precision	recall	f1-score	support
negative	0.89	0.86	0.87	5090
positive	0.86	0.89	0.87	4910
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000



Conclusions

In summary, this project aimed to perform sentiment analysis on IMDb movie reviews using a diverse set of classifiers such as MultinomialNB, LinearSVC, LogisticRegression, and SGDClassifiers. The methodology involved preprocessing with regex and NLTK and creating a Bag-of-Words using scikit-learn. The classifiers were trained and evaluated on a dataset split into training and test sets, with performance metrics such as accuracy, precision, recall, f1-score and confusion matrices analyzed using scikit-learn tools. The results demonstrated the effectiveness of the models in classifying sentiments within movie reviews. Moving forward, potential future works could involve exploring more advanced tokenization techniques, experimenting with alternative methods to extract the features (i.e. TF-IDF) and incorporating deep learning models for improved sentiment analysis.

Bibliography

[1] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding". 2020.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". 2019.