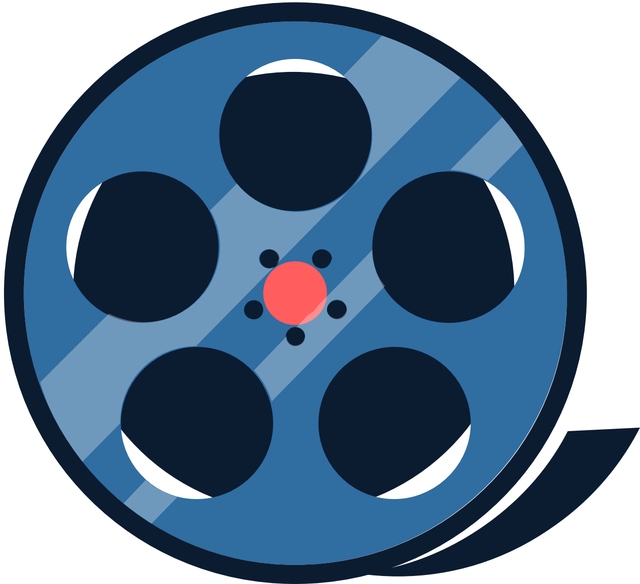
IMDb Movies Dataset

Introduction

IMDb Movies is the ultimate destination for movie lovers around the world. With a vast database of over 7 million titles, it is the go-to source for information on movies, TV shows, and celebrities. From classic films to the latest blockbusters, IMDb Movies has everything you need to stay up to date on your favorite flicks. Whether you're looking for reviews, ratings, trailers, or showtimes, IMDb Movies has got you covered.

Problem Description

The problem with IMDb Movies is that the reviews provided by users are often subjective and can vary greatly in quality, making it difficult for other users to determine the overall sentiment of a movie. This inconsistency in reviews can lead to confusion and frustration for those trying to make informed decisions about which movies to watch. To solve the problem, we can classify reviews into positive or negative.



Method

1

Naive byes

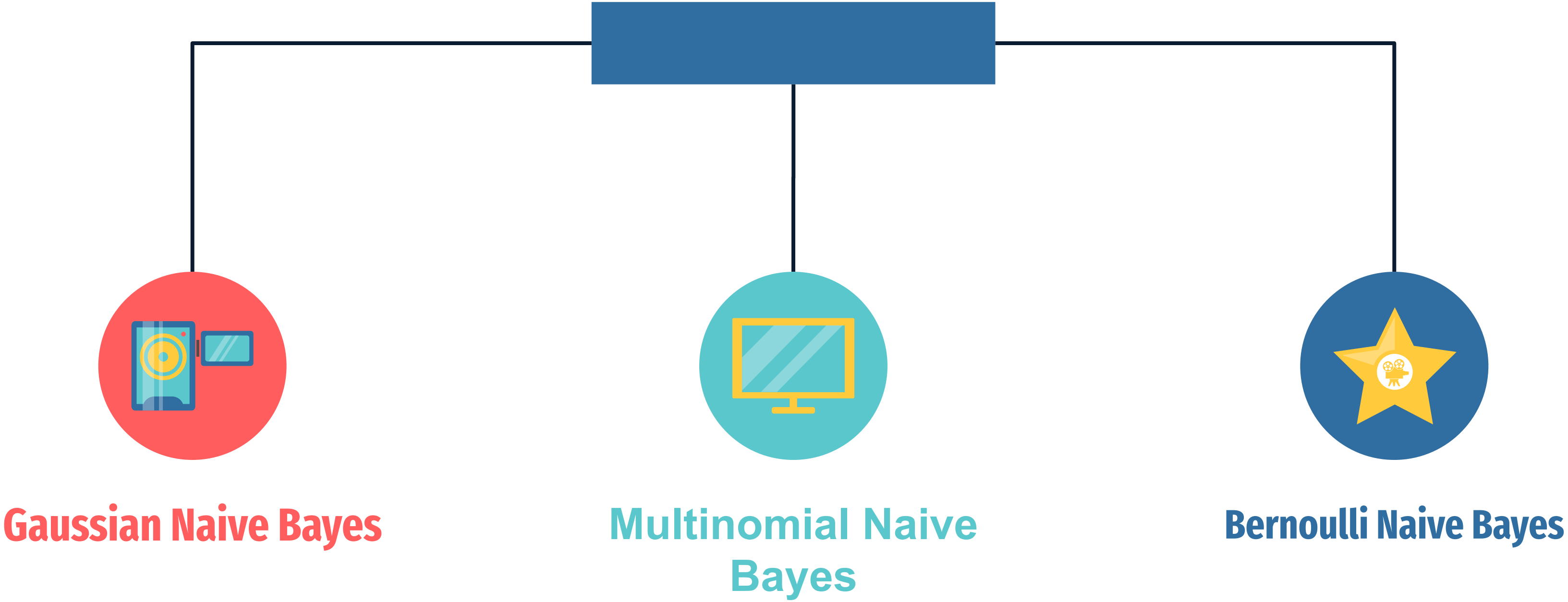
2

SVM

Naive Bayes

Naive Bayes is a probabilistic classification algorithm that is based on Bayes' theorem. It is called "naive" because it makes a simplifying assumption that all features in the dataset are independent of each other, given the class variable. This assumption makes the algorithm computationally efficient and easy to implement

There are three main types of Naive Bayes classifiers:



Split the data into training and testing sets

```
# Split the data into training and testing sets
X = movie_reviews.review  # Input features
y = movie_reviews.sentiment  # Target variable

# Split the data into 80% training and 20% testing, using random_state for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Print the shape of the training and testing sets
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
print("Training labels shape:", y_train.shape)
print("Testing labels shape:", y_test.shape)
```

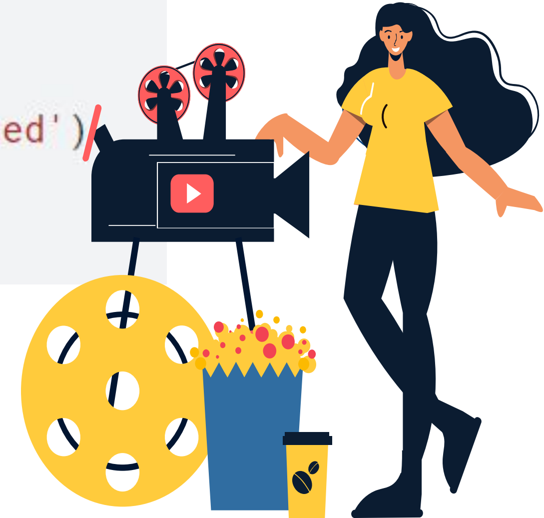
```
Training data shape: (40000,)
Testing data shape: (10000,)
Training labels shape: (40000,)
Testing labels shape: (10000,)
```

Train the model and Make predictions on the test data, evaluation metrics

```
# Train the model
classifier = MultinomialNB()
classifier.fit(X_train_transformed, y_train)

# Make predictions on the test data
y_pred = classifier.predict(X_test_transformed)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
```



confusion matrix

```
# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)

# Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)
```

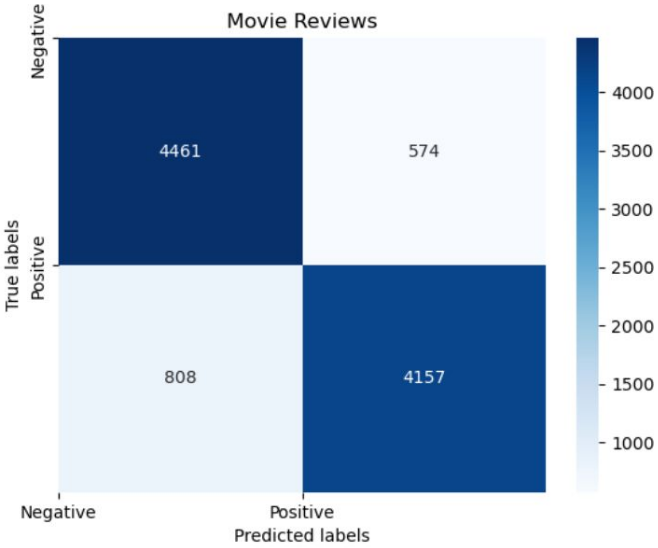
```
Accuracy: 0.8618
Precision: 0.8625493233835743
Recall: 0.8618
F1-score: 0.8617015991098335
Confusion matrix:
[[4461  574]
 [ 808 4157]]
```

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
#Create a confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Create a heatmap for the confusion matrix
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')

# Add labels, title, and axis ticks
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Movie Reviews')
plt.xticks(ticks=[0, 1], labels=['Negative', 'Positive'])
plt.yticks(ticks=[0, 1], labels=['Negative', 'Positive'])

# Show the plot
plt.show()
```



Advantages

1. Simple and easy to implement

2. highly scalable with the number of predictors and data points

3. Fast training and prediction and high-dimensional data

4. fast and can be used to make real-time predictions

Disadvantages

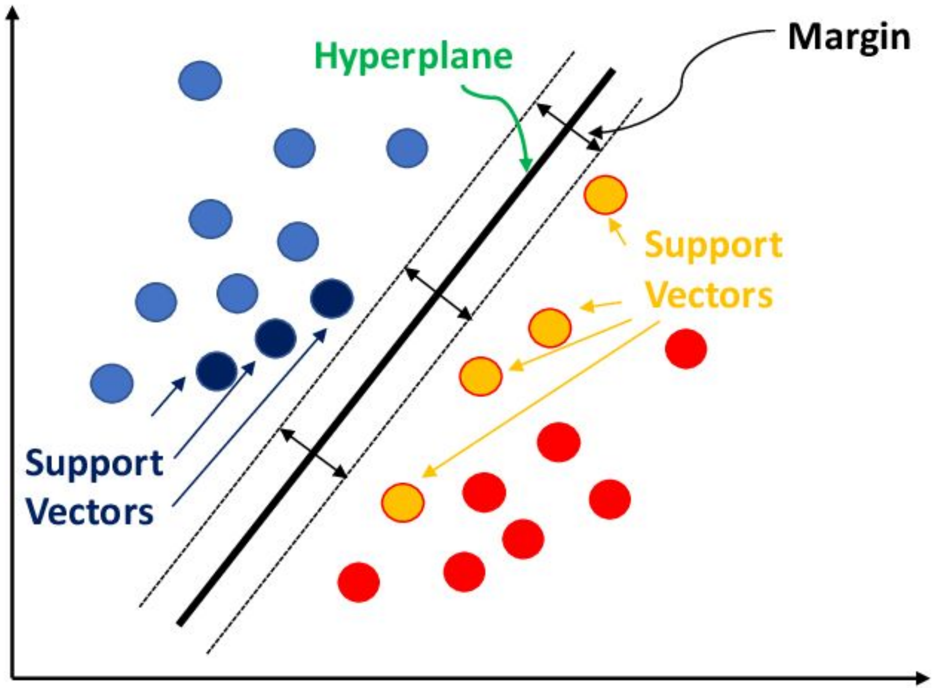
1. Naive Bayes assumes that all predictors (or features) are independent
2. estimations can be wrong in some cases
3. This algorithm faces the ‘zero-frequency problem’
4. Biased towards categorical features:

Support Vector Machines

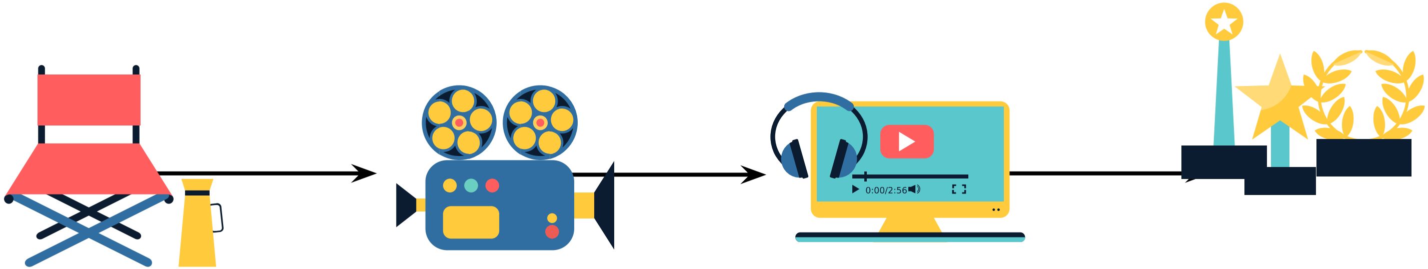
SVMs were initially introduced in the 1960s but were later developed in 1990. SVMs are implemented differently from other machine learning algorithms. They have recently gained popularity due to their capacity to manage numerous continuous and categorical variables.

An SVM model represents different classes in a hyperplane in multidimensional space. The hyperplane will be generated iteratively by SVM to minimize the error. SVM aims to divide the datasets into classes to find a maximum marginal hyperplane (MMH).

SVM



Important concepts in support vector machines



1 Support Vectors

the data points closest to the hyperplane. These data points will be used to define the dividing line

2 Hyperplane

is a decision plane or space that is divided between a collection of objects with various class designations.

3 Margin

The distance between two lines on the closet data points of various classes can be used to define margin.

The perpendicular distance between the line and the support vectors can be used to calculate it. A large margin is considered good, whereas a small margin is considered bad.

Advantages

SVM classifiers perform well in high-dimensional space and have excellent accuracy. SVM classifiers require less memory because they only use a portion of the training data

- SVM performs reasonably well when there is a large gap between classes.

- High-dimensional spaces are better suited for SVM.

- SVM uses memory effectively.

Disadvateges

Large data sets are not a good fit for the SVM algorithm

- SVM requires a long training period; as a result, it is not practical for large datasets.

- When the data set contains more noise, such as overlapping target classes, SVM does not perform as well.

- The inability of SVM classifiers to handle overlapping

Support Vector Machine Model

```
#Import svm model
from sklearn import svm

#Creating a svm Classifier
svm = svm.SVC(kernel='linear')

#Training the model
svm.fit(train_x_vector , y_train)
```

```
SVC
SVC(kernel='linear')
```

```
[ ] #Predict the response for test dataset
y_pred = svm.predict(test_x_vector)
```

Make predictions on the test data

```
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8911
```

```
#Precision
print("Precision:",metrics.precision_score(y_test, y_pred))

#Recal
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Precision: 0.8840665873959572
Recall: 0.8984894259818731
```

Hyperparameter Tuning

```
[ ]  from sklearn.svm import SVC
     from sklearn.model_selection import GridSearchCV

     param_grid = {'C': [0.1, 1, 10, 100, 1000],
                   'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                   'kernel': ['rbf']}

     grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
```

Conclusion

The use of IMDb Movies sentiment analysis dataset provides insights into how people perceive and react to movies. This information can be useful for movie studios and filmmakers to understand audience reactions to their work and make adjustments to improve future projects.