```
# %%
import numpy as np
import pandas as pd
from collections import Counter
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.model_selection import train_test_split
from sklearn.utils.validation import check_X_y, check_array, check_is_fitted
from sklearn.utils.multiclass import unique_labels
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import euclidean_distances, accuracy_score,
classification_report

# %%
class NaiveBayesClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, alpha=1):
        self.alpha = alpha

    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)
        self.X_ = X
        self.y_ = y
        self.n_samples, self.n_features = X.shape
        self.n_classes = len(self.classes_)
        self.prior = np.zeros(self.n_classes)
        self.likelihood = np.zeros((self.n_classes, self.n_features))
        self._calculate_prior()
        self._calculate_likelihood()
        return self

    def _calculate_prior(self):
        for i, c in enumerate(self.classes_):
            self.prior[i] = np.sum(self.y_ == c) / self.n_samples

    def _calculate_likelihood(self):
        for i, c in enumerate(self.classes_):
            self.likelihood[i] = (np.sum(self.X_[self.y_ == c], axis=0) +
self.alpha) / (np.sum(self.X_[self.y_ == c]) + self.alpha * self.n_features)

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])
        # Input validation
        X = check_array(X)
        return np.array([self._predict(x) for x in X])

    def _predict(self, x):
        posteriors = np.zeros(self.n_classes)
        for i, c in enumerate(self.classes_):
            posteriors[i] = np.log(self.prior[i]) +
np.sum(np.log(self.likelihood[i]) * x)
        return self.classes_[np.argmax(posteriors)]
```

```
# %%
base = pd.read_csv('re8.csv')
X = base['text']
y = base['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# %%
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test_transformed = vectorizer.transform(X_test)

# %%
clf = NaiveBayesClassifier()
clf.fit(X_train.toarray(), y_train)

# %%
predictions = clf.predict(X_test_transformed.toarray())
accuracy = accuracy_score(y_test, predictions)
cp = classification_report(y_test, predictions)

print('Accuracy:', accuracy)
print(cp)
```