```
#!pip install gensim scikit-learn datasets
```

Load data

```
from datasets import load_dataset

dataset = load_dataset("zeroshot/twitter-financial-news-sentiment")
train_dataset = dataset['train']
test_dataset = dataset['validation']

print(f"Training dataset size: {len(train_dataset)}")
print(f"Validation dataset size: {len(test_dataset)}")
```

```
Training dataset size: 9543
Validation dataset size: 2388
```

Implement textclassifier

```
from sklearn.feature_extraction.text import TfidfVectorizer as SklearnTfidfVectorizer

class Vectorizer:
    def fit_transform(self, documents):
        raise NotImplementedError("Subclass must implement abstract method")

class TfidfVectorizer(Vectorizer):
    def __init__(self, **kwargs):
        self.vectorizer = SklearnTfidfVectorizer(**kwargs)

    def fit_transform(self, documents):
        return self.vectorizer.fit_transform(documents)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

class TextClassifier:
    """Text classifier using Logistic Regression."""
    def __init__(self):
        self.model = LogisticRegression()

    def fit(self, X_train, y_train):
        """Fits the Logistic Regression model."""
        self.model.fit(X_train, y_train)

    def predict(self, X_test):
        """Makes predictions using the fitted model."""
        return self.model.predict(X_test)

    def evaluate(self, y_true, y_pred):
        """Prints the classification report."""
        print(classification_report(y_true, y_pred))
```

Train and evaluate

```
train_texts = [item['text'] for item in train_dataset]
train_labels = [item['label'] for item in train_dataset]
val_texts = [item['text'] for item in test_dataset]
val_labels = [item['label'] for item in test_dataset]

vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(train_texts)
X_val = vectorizer.vectorizer.transform(val_texts) # Use transform on validation data

classifier = TextClassifier()
classifier.fit(X_train, train_labels)
```

```
predictions = classifier.predict(X_val)
classifier.evaluate(val_labels, predictions)
```

```
              precision    recall  f1-score   support

           0       0.79      0.44      0.56       347
           1       0.77      0.56      0.65       475
           2       0.81      0.96      0.88      1566

    accuracy                           0.80      2388
   macro avg       0.79      0.65      0.70      2388
weighted avg       0.80      0.80      0.79      2388
```

Improve preprocessing and feature selection

Apply noise filtering and vocabulary reduction techniques to the text data.

```
import re
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import nltk

# Download necessary NLTK data
nltk.download('stopwords')
nltk.download('wordnet')

def preprocess_text(documents):
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))
    processed_docs = []
    for doc in documents:
        # to lowercase + remove punctuation, num, stopwords + apply lemmatization
        doc = doc.lower()
        doc = doc.translate(str.maketrans('', '', string.punctuation))
        doc = re.sub(r'\d+', '', doc)
        words = doc.split()
        words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
        processed_docs.append(' '.join(words))
    return processed_docs

train_texts_processed = preprocess_text(train_texts)
val_texts_processed = preprocess_text(val_texts)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
new_vectorizer = TfidfVectorizer(max_df=0.95, min_df=5, max_features=10000)
X_train_processed = new_vectorizer.fit_transform(train_texts_processed)
X_val_processed = new_vectorizer.vectorizer.transform(val_texts_processed)
```

```
#Experiment with more complex model architectures
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier

nb_model = MultinomialNB()
gb_model = GradientBoostingClassifier()
mlp_model = MLPClassifier(max_iter=1000)
```

```
print("Evaluating Multinomial Naive Bayes Model:")
nb_model.fit(X_train_processed, train_labels)
nb_predictions = nb_model.predict(X_val_processed)
print(classification_report(val_labels, nb_predictions))

print("\nEvaluating Gradient Boosting Classifier Model:")
gb_model.fit(X_train_processed, train_labels)
gb_predictions = gb_model.predict(X_val_processed)
print(classification_report(val_labels, gb_predictions))

print("\nEvaluating MLP Classifier Model:")
```

```
mlp_model.fit(X_train_processed, train_labels)
mlp_predictions = mlp_model.predict(X_val_processed)
print(classification_report(val_labels, mlp_predictions))
```

```
Evaluating Multinomial Naive Bayes Model:
              precision    recall  f1-score   support

           0       0.79      0.27      0.40       347
           1       0.72      0.48      0.58       475
           2       0.77      0.97      0.86      1566

    accuracy                           0.77      2388
   macro avg       0.76      0.57      0.61      2388
weighted avg       0.77      0.77      0.74      2388


Evaluating Gradient Boosting Classifier Model:
              precision    recall  f1-score   support

           0       0.85      0.25      0.39       347
           1       0.86      0.35      0.49       475
           2       0.74      0.99      0.84      1566

    accuracy                           0.75      2388
   macro avg       0.82      0.53      0.58      2388
weighted avg       0.78      0.75      0.71      2388


Evaluating MLP Classifier Model:
              precision    recall  f1-score   support

           0       0.57      0.57      0.57       347
           1       0.63      0.65      0.64       475
           2       0.84      0.84      0.84      1566

    accuracy                           0.76      2388
   macro avg       0.68      0.68      0.68      2388
weighted avg       0.76      0.76      0.76      2388
```