I chose support vector machines (LinearSVMs) and gradient booster (xgboost) as classifiers for this task.

1. My text preprocessing pipeline starts with text cleaning. This includes normalizing the texts (casting to ascii by removing accents e.g. ‘ä’ ⟶ ‘a’), lowercasing, leaving only latin letters separated by spaces (I also kept “‘“ symbol that is used in word contractions, e.g. “don’t”).

   Then, my pipeline splits into 2 parts:
   - One part is tokenizing, lemmatizing and vectorizing the texts. I used *SpaCy* package for tokenization and lemmatization. During tokenization and lemmatization, I additionally:
     - joined negations with the words that are negated (e.g. “isn’t good” ⟶ “NOT_good”);
     - removed articles, single character words, and lemmas containing “‘“ (usually leftover from other words, e.g. “‘s”).

     The vectorizer included ngrams, and used only top 6k or 1k features (see ‘*Differences on vectorizer choice’*).
   - Another part was calculating a total embedding of each review using word embeddings. I used word vectors supplied by *SpaCy* to find an embedding of each word, and then summing them over, resulting in a total embedding of a whole review. The embeddings are 300 dimensional.

   Both of these approaches were concatenated (using *scikit-learn*’s FeatureUnion).

   Differences on vectorizer choice for SVM vs xgboost classifier.
   SVMs perform best when features are normalized/scaled, so I chose tf-idf vectorizer and also applied StandardScaler on the calculated total embeddings. Tf-idf vectorizer used 3-grams and top 6000 features.
   Decision trees do not need normalized data, so I chose CountVectorizer (seemed to outperform tf-idf) and didn’t apply scaling on total embeddings. Also, to try and reduce sparsity, I chose to use 2-grams and only top 1000 features.

2. I chose SVMs, because they provide robust performance when dealing with high-dimensional, sparse feature space, when the dataset size is moderate. This applies to the provided text dataset with ~10k samples, each containing up to 300 words (even less unique words), while the vocabulary of the whole dataset is >6000 (even more when including ngrams).
   On the other hand, decision trees are not supposed to be good at dealing with high-dimensional sparse data, but xgboost classifier was proven to be a state-of-the-art classifier that performed well in a number of Kaggle competitions.

3. I split the data into training, validation, and test sets. Preprocessing steps and model parameters were tuned using training and validation sets. Once I was happy with the models, I evaluated them on the test set using accuracy as a metric. SVM performed slightly better both on validation and on test sets, with 79.4% mean cross-val accuracy and 78.2% test accuracy. Xgboost mean cross-val accuracy was 76.4% and test set accuracy was 76.5%.
   Overall, both models give comparable results and cost about the same to predict (the text preprocessing step is the expensive one). However I prefer SVMs because from personal experience they are good, robust models for text classification.

4. If the dataset was imbalanced, accuracy would not be the best metric, as it is defined as all_correct_predictions/number_of_samples, thus it would be enough to always guess the larger class to get good accuracy (the higher the imbalance, the higher the effect). Precision and recall would be better in this case, as they would tell us what percentage of the predictions for the class are correct predictions (precision), and what percentage of relevant predictions we are able to find (recall).

Bonus: I have experimented with some bidirectional LSTM architectures with GloVe embeddings. The validation accuracy achieved with rough tuning was 76.6% (and 75% on the test set). Concerning underperformance relative to SVMs, on the one hand the length of reviews is very reasonable for LSTMs, on the other hand, the dataset size is most likely insufficient for this network and/or radical and careful regularization would need to be applied.