Write-up of the methods I used in the competition

The organizers of the competition set a challenge: to classify Amazon reviews with the highest accuracy. An important note is that there are artificially created label issues in the training data.
Accuracy is taken as the metric. In this article I will explain how I managed to get 88% quality on the public leaderboard (Rank 3) and 87.33% on the private leaderboard.

The main idea of the work was to go from simple to complex.

- From simple ways of pre-processing data such as TFIDF to more complex ones such as embeddings.
- From simple models, such as logistic regression and publicly available pretrained models, to more complex ones, such as neural networks and model ensembles.
- From testing models on a "dirty" set to gradually cleaning it up and testing models on a "clean" set

Globally, the chapters of this paper are broken down according to the features on which the models were trained and tested. A total of four approaches were used to preprocess the features:

- TFIDF
- Universal Sentence Encoder
- Universal Sentence Encoder Large
- Universal Sentence Encoder CMLM

Subchapters of this work reflect what models were tested on these signs.

Before moving on to the main body of the work, the data was explored and 3 new features were created:
words_only. Words extracted from the original text review_text. Without numbers, symbols or punctuation marks.
lemmas. Lemmas extracted from the original review_text. Contains words-lemmas and numbers, without punctuation symbols.
words_count. Number of words in a sentence.
From which words_only and lemmas were used in further experiments.

PART 1: SIMPLE PREPROCESSING

CHAPTER 1. TF-IDF FEATURES

WordsOnly + TFIDF + SGDClassifier

First we took *WordsOnly* feature and applied TFIDF transformation to it. After that we checked the quality of SGDClassifier model from Scikit-learn library. First we have tested the quality of the model on 5-fold cross validation, then we have selected hyperparameters and tested the quality of the model prediction with the selected hyperparameters in a public test. The results of the experiments are presented below:

	Train (5-fold-CV)	Public Leaderboard
Default	0.68471	-
Tuned	0.68511	0.75333

Lemmas + TFIDF + SGDClassifier

Next, we applied TFIDF transofrmation to the Lemmas feature. TFIDF fitted on lemmas performed slightly worse on cross validation but showed a bit better result on public leaderboard.

	Train (5-fold-CV)	Public Leaderboard
Default	0.67389	-
Tuned	0.67444	0.76667

At this stage the hyperparameters for the SGDClassifer model were found:

- alpha: 0.0001
- l1_ratio: 0.2
- penalty: elasticnet

Hereinafter the model with such hyperparameters will be called BestSGDClassifier for convenience.

For TFIDF transformer we also selected the most optimal hyperparameters:

- max_df: 0.5

In the future this hyperparameter will be set for TFIDF transormer while working with BestSGDClassifer model.

Lemmas + TFIDF + BestSGDClassifier + CleanlabLabels(by BestSGDClassifier)

For further experiments we decided to use a logistic regression model with hyperparameters that were found in the previous step - BestSGDClassifier.

Let's use it to find issue labels and check its quality on cross-validation and public test data. We use the CleanLab library to search for issue labels.

	Train (5-fold-CV)	Public Leaderboard
BestSGDClassifier	0.84741	0.72667

However, in spite of the higher score on cross-validation, the quality on the public test data was even worse than that of the same model trained on the whole dataset.

Apparently, the data cleaning was not of sufficient quality. This leads to the question, which model can be used to find the highest number of valid label issues? Let's try the BERT model.

Lemmas + TFIDF + BestSGDClasssifer + CleanlabLabels(model=BERT)

So, to clean the data from label issues, to create high-quality predicted probabilities, which will later be passed to a special CleanLab library method, it was decided to use the pre-trained BERT model. This model was specially trained to classify Amazon reviews (https://huggingface.co/LiYuan/amazon-review-sentiment-analysis). This model was not pre-trained on the current data, as it was not necessary.
The quality that the pre-trained BERT showed on the training data was 0.61697, but the accuracy of the model on the public test data was 0.81333, which is very good. So far, it outperforms the quality of all the models we have tested so far. This difference between the quality on the training and test data could only indicate that there are indeed label issues in the data.
So, we use BERT and CleanLab to get rid of label issues and check BestSGDClasssifer again.

	Train(drop all issues) (10-fold-CV)	Public Leaderboard
BestSGDClassifier	0.80023	0.77333

After such data cleaning, BestSGDClassifier quality on cross validation was 0.80023, and accuracy on public test data was 0.77333. At the moment, this is the best performance we have received. Conclusion: the BERT model was able to give high-quality predicted probabilities, which, in combination with the Cleanlab library, we were able to clean the data. In the future, we will exclude from the data the rows that were found at this stage.

PART 2: EMBEDDINGS

In this part of the work, we will train models on embedding. We will use three models to create embeddings:
- Universal Sentence Encoder (USE)
- Universal Sentence Encoder Large (USE LARGE)
- Universal Sentence Encoder CMLM (USE CMLM)

CHAPTER 1: USE EMBEDDINGS

The Universal Sentence Encoder encodes text into high-dimensional vectors that can be used for text classification, semantic similarity, clustering and other natural language tasks. The model is trained and optimized for greater-than-word length text, such as sentences, phrases or short paragraphs. It is trained on a variety of data sources and a variety of tasks with the aim of dynamically accommodating a wide variety of natural language understanding tasks. The input is variable length English text and the output is a 512 dimensional vector. The universal-sentence-encoder model is trained with a deep averaging network (DAN) encoder.

LogReg(solver=lbfgs, iter=1000) | BestSGDClassifier + CleanlabLabels(model=BERT) | also finding best fraction of droppped issues_idx

In this part of the work we will train two logistic regression models and test their quality on the public dataset, and also find the optimal number of label issues that should be excluded from the training dataset. The quality will be tested on public test data.
Embeddings obtained from the USE model will be used for training. The size of the embedding is 512 features.
So first of all let's check the results of several models on 10-fold cross validation:

- LogReg(solver=lbfgs, iter=1000))
- LogReg(solver=liblinear)
- SGDClassifier (default)
- BestSGDClassifier

Then the quality of the model with the best cross-validation result will be tested on a public test.
As a result of this experiment, the logistic regression model LogReg(solver=lbfgs, iter=1000) showed the best result on cross-validation (0.78235), then it was checked on the test and showed the quality 0.77333. The results of all other models are presented below:

	Train(drop all issues) (10-fold-CV)	Public Leaderboard
LogReg(solver=lbfgs, iter=1000)	0.78235	0.77333
LogReg(solver=liblinear)	0.77531	-
SGDClassifier (default)	0.773675	-
BestSGDClassifier	0.77184	0.79333

The next step was to test how the quality of the model would change on the public test, depending on the number of deleted label issues. The test was carried out and from 500 to

3000 label issues were removed from the training data. As a result we got the following results:

- drop-500 - 0.76667
- drop-1000 - 0.76667
- drop-1500 - 0.77333
- **drop-2000 - 0.78000**
- drop-2500 - 0.77333
- drop_all - 0.77333

The LogReg(solver=lbfgs, iter=1000) model showed the best result in the public test after training on data from which 2000 label issues were removed.

The next step was to test the BestSGDClassifier model (remember, this is the SGDClassifier model with the hyperparameters selected earlier) in a public test. As a result, an even higher quality was obtained: 0.79333.
Then from 500 to 3000 instances of label issues were also sequentially removed from the training sample with a step of 500, the BestSGDClassifier model was trained on each data set, and after each training the predictions were made on the public test. The final results were as follows:

- drop-500 - 0.77333
- drop-1000 - 0.78000
- drop-1500 - 0.77333
- drop-2000 - 0.80667
- drop-2500 - 0.81333
- drop_all 0.79333

The BestSGDClassifier model showed the best result after training on the training data set, from which 2500 label issues were removed. Moreover, the result the model showed on 10-fold cross-validation on such dataset was 0.93568.

For further experiments it was decided to take a training sample with the number of removed label issues of 2500.

A further experiment also attempted to find hyperparameters for the SGDClassifier model on a new cleaned dataset, but the results that were obtained (5-fold-cv best = 0.75886, public = 0.80000) could not outperform the BestSGDClassifier model.

OVR, OVO, OCC + CleanlabLabels(model=BERT)[:2500]

What if we try to fit the models to each class separately?
The next step is to test the work of three models of multiclass classification:

- OneVsRestClassifier (OVR)
- OneVsOneClassifier (OVO)
- OutputCodeClassifier (OCO).

The following results were obtained as a result of the test.

	Train(drop top 2500 issues) (10-fold-CV)	Public Leaderboard
OVR	0.75680	0.80667
OVO	0.76675	0.76667
OCC	0.73818	0.76667

None of the models showed a significant increase in quality, so it was decided to refrain from using them for the further stages of the experiments.

MEAN(BestSGDClassifier & BERT) + CleanlabLabels(model=BERT)[:2500]

One effective way to ensemble models in machine learning is to average their predictions. Let's try to take the two models that showed the best quality on the public test: BERT and BestSGDClassifier (trained on training data with deleted top-2500 label issues), make predictions on the public test, and average them.
We use the predicted logits of the BERT model and the decision function BestSGDClassifier as predictions for averaging.
As a result, the prediction quality on the public test has reached a value of 0.83333. This is the best quality score obtained so far.

3-layer-NN + CleanlabLabels(model=BERT)[:2500]

Then the work of the three-layer neural network was tested. The quality of its performance was:
Validation accuracy: 0.7667
Public test accuracy: 0.82000

CHAPTER 2: USE LARGE

Then, it was decided to use the USE LARGE model to create embeddings.

BestSGDClassifier + CleanlabLabels(model=BERT)[:2500]

First of all, let's check the model that showed one of the best results in previous experiments: the BestSGDClassifer model. The quality of its performance is shown in the table below:

	Train(drop top 2500 issues) (10-fold-CV)	Public Leaderboard
BestSGDClassifer	0.79403	0.84000

Compare with the results that were obtained by the same model, but on embeddings obtained with the light version of USE:

	Train(drop top 2500 issues) (10-fold-CV)	Public Leaderboard
BestSGDClassifer	0.76271	0.81333

Как вы можете заметить, одна и та же модель показала значительно лучший результат на кросс-валидации и на публичном тесте, будучи обученной на новых эмбеддингах.

MEAN(BestSGDClassifier & BERT) + CleanlabLabels(model=BERT)[:2500]

The next step is to check again the combination that showed one of the best results in the previous experiments: that is, we add the logits obtained from the BERT model and the decision function obtained from the BestSGDClassifier with a coefficient of 0.5 for each. However, this time the BestSGDClassifier model was trained on the embeddings obtained from the USE LARGE model.
The quality on the public test was 0.87333. This is a significant increase in quality compared to the same combination of models when BestSGDClassifier was trained on old embeddings.

The next step was to try to vary the coefficients in the model predictions between 0 and 1. The results are presented below:

- 0.0*BERT_logits + 1.0*BestSGDClassifier_decision_func - 0.84000 (only BestSGDClassifier)
- 0.1*BERT_logits + 0.9*BestSGDClassifier_decision_func - 0.84667
- 0.2*BERT_logits + 0.8*BestSGDClassifier_decision_func - 0.85333
- 0.3*BERT_logits + 0.7*BestSGDClassifier_decision_func - 0.85333
- 0.4*BERT_logits + 0.6*BestSGDClassifier_decision_func - 0.86667
- 0.5*BERT_logits + 0.5*BestSGDClassifier_decision_func - 0.87333
- 0.6*BERT_logits + 0.4*BestSGDClassifier_decision_func - 0.86000
- 0.7*BERT_logits + 0.3*BestSGDClassifier_decision_func - 0.84667
- 0.8*BERT_logits + 0.2*BestSGDClassifier_decision_func - 0.84667

- 0.9*BERT_logits + 0.1*BestSGDClassifier_decision_func - 0.83333
- 1.0*BERT_logits + 0.0*BestSGDClassifier_decision_func - 0.81333 (only BERT)

The models with prediction coefficients equal to 0.5 showed the best quality.

SGDClassifier_GridSearchCV + CleanlabLabels(model=BERT)[:2500]

The next step was again an attempt to find hyperparameters for the SGDClassifier model. The selection was performed on the embeddings obtained using the USE LARGE model with the top-2500 label issues removed. However, this fitting did not help to improve the quality neither on the cross-validation nor on the public test:
The results are presented below:

	Train (drop top 2500 issues) (5-fold-CV)	Public Leaderboard
Default	0.78516	-
Tuned	0.78861	0.80667

Then the next step was to take the entire sample, without dropping label issues.
As a result, almost all the same hyperparameters were chosen as for the BestSGDClassfier model, except that this time the model was slightly more inclined towards L1-regularization (the coefficient increased from 0.2 to 0.5). The model with the obtained hyperparameters was then trained on a training sample cleaned of top-2500 label issues. The quality of such a model on the public test was 0.84000.
The results of the fitting are presented below.

	Train full dataset (5-fold-CV)	Public Leaderboard
Default	0.66701	-
Tuned	0.67223	0.84000

It was decided to keep the BestSGDClassifier model as a linear model for further experiments.

KNN + CleanlabLabels(model=BERT)[:2500]

For the sake of curiosity, it was also decided to test the quality of the KNN classifier. The results of the test are presented below:

	Train(drop top 2500 issues) (10-fold-CV)	Public Leaderboard
KNN Classifier	0.71602	0.75333

Well, the quality of the KNN classifier left much to be desired.

CHAPTER 3: USE CMLM

The next step was to experiment with the embeddings obtained by the Universal sentence encoder for English trained with conditional masked language model. This is a model that map text into high dimensional vectors that capture sentence-level semantics. It is an English-large (en-large) model which is trained using a conditional masked language model. The model is intended to be used for text classification, text clustering, semantic textural similarity, etc. It can also be use used as modularized input for multimodal tasks with text as a feature. The large model employs a 24 layer BERT transformer architecture. This model returns a vector with a dimension of 1024 attributes.

BestSGDClassifier + CleanlabLabels(model=BERT)[:2500]

First of all, let's check on the new embeddings the performance of a model that has proven itself in previous experiments: BestSGDClassifier.

	Train(drop top 2500 issues) (10-fold-CV)	Public Leaderboard
KNN Classifier	0.78684	0.74000

In this case, the model was trained with 1024 dimensions, and the quality was much worse than when it was trained with 512 embeddings.
Perhaps these embeddings are not well suited for linear models. Let's try to use them to train a neural network.

NN + CleanlabLabels(model=BERT)[:2500]

The next step was to test the performance of several neural networks that were slightly different from each other.
The best result was shown by a three-layer neural network with the following architecture:

Input layer (shape=(1024,)
Denselayer(1024)
BatchNormalization()
ReLU()
Dropout (rate=0.3)
_____
Dense layer (shape=512)
BatchNormalization()
ReLU()
Dropout(rate=0.3, seed=RANDOM_SEED),
_____
Dense(shape=256),
BatchNormalization()
ReLU()
Dropout(rate=0.3, seed=RANDOM_SEED)
Dense output layer (shape=5)

The Adam algorithm was used as the optimization algorithm, with the learning_rate parameter=0.0001.

The quality of the model performance is presented below:

	Train(drop top 2500 issues) (10-fold-CV)	Public Leaderboard (4 epoch)	Public Leaderboard (5 epoch)
NN	0.82269	0.79333	0.83333

To get results on cross validation, the model was trained 4 epochs, to get results on test data, 2 models were tested: one trained 4 epochs, one trained 5. The last one showed the best quality.
Hereinafter for simplicity we will call this model BestNN (Best Neural Network)

MEAN([*USE_LARGE*/BestSGDClassifier] & [BERT] & [*USE_CMLM*/BestNN])) + CleanlabLabels(model=BERT)[:2500]

The next step was to average the predictions of the different models and test the quality of this averaging in a public test.
The three models which predictions were used in the experiment:

- BestNN
- BestSGDClassifier
- BERT

Two kinds of predictions were averaged:

- Real predictions (such as logits and decision function values)
- Predicted probabilities.

The quality of the models in the public test are shown below:

	Real value predictions	Predicted probabilities
MEAN(BestNN_preds + BestSGDClassifier_preds + BERT_preds)	0.86000	0.86000
MEAN(BestNN_preds + BestSGDClassifier_preds)	0.84667	0.86000
MEAN(BestNN_preds + BERT_preds)	0.83333	0.84000

None of the methods showed higher quality than what was achieved by averaging the real predictions of the BERT and BestSGDClassifier models.

QUICK SUMMARY

- USE LARGE Embeddings are good for logistic regression.
- USE CMLM embeddings with higher dimensionality are good for neural network performance.
- Label issues, which removal allowed the logistic regression models to show the best results after training on a new cleaned sample, were found using the CleanLab library and class probabilities predicted by the pre-trained BERT model.
- The logistic regression model with selected hyperparameters showed the best performance on the public test after removing top-2500 label issues from the training data.
- The following hyperparameters proved to be the best for logistic regression trained with stochastic gradient descent:
    - alpha: 0.0001
    - l1_ratio: 0.2
    - penalty=*elasticnet*
- The best quality on the public test was obtained by averaging the logits predicted by the BERT model and the decision function obtained from the BestSGDClassifier model.

PART 3. STACKING

At this stage, we will try to combine the predictions from the BestNN and BestSGDClassifer models and train a metamodel on them. As a metamodel we will use the model of gradient boosting from the CatBoost library.

The quality that we were able to obtain on cross-validation and on the public test is presented below:

	Train(drop top 2500 issues) (10-fold-CV)	Public Leaderboard
Metamodel	0.82309	0.86667

This approach did not help to improve the quality of the model performance on the public test.
Remember that when averaging with the predictions from the pre-trained BERT model, our models previously gave better quality. Let us try to average the real predictions of the metamodel and the logits of the BERT model. Let us test the quality of this approach immediately on a test dataset:

	Public Leaderboard
MEAN(Metamodel_preds + BERT_preds) (Real value predictions)	0.88000

Due to this averaging we were able to increase the quality of prediction on the public test up to 88%! At the moment, this is the best score!

We'll use this approach as the final solution.

PART 4: FINAL SUBMISSION

At this point, we will use the approach that gave us the highest quality on the public test, and use it to create predictions for the private test. For this purpose let's create the following predictions:

- BestSGDClassifier decision function predictions (preliminarily created embeddings using USE LARGE model on the test data)
- BestNN predicted logits (preliminarily created embeddings using USE CMLM model)
- BERT predicted logits

Next, we will combine the predictions from the BestSGDClassifier and BestNN models, and make predictions using the metamodel on the resulting features. Thus we will get one more vector of predictions:

- Metamodel predicted raw real values

Then let's average the predictions of the Metamodel and BERT. That will be our final vector of predictions for the private test. The quality of these predictions was 84% on the private test data set.

However the organizers gave us a chance to make some submits, the best of them will be shown on the private leaderboard. Well, let's experiment and see what quality of each model's predictions separately and try to improve the current result a little bit more.

Below you can see the quality of the predictions from the intermediate models:

	Private Leaderboard
BestNN predictions	0.76333
BestSGDClassifier predictions	0.79333
BERT predictions	0.81333
Metamodel predictions	0.78667

Let's try, for the sake of curiosity, to mix the different predictions with each other, using different coefficients:

Sometimes a simple solution is the best solution. Remember that during the experiments, one of the best combinations of models was BERT and BestSGDClassifier. Let us combine the real predictions of these models and check the quality:

	Private Leaderboard
MEAN(BERT & BestSGDClassifier predictions)	0.85333

Surprisingly, this combination really improved the final quality. Let's not stop and try to add metamodel predictions:

	Private Leaderboard
MEAN(BERT & BestSGDClassifier & Metamodel predictions)	0.85333

The quality remained the same. From past experience, when we added the predictions from BERT, the prediction quality improved. Let's try to increase the coefficient in front of the BERT model predictions. Let's see how the quality changes:

	Private Leaderboard
MEAN(2xBERT & BestSGDClassifier & Metamodel predictions)	0.86000

Bingo! The hypothesis worked. Let's keep moving in that direction and increase the coefficient by one more:

	Private Leaderboard
MEAN(3xBERT & BestSGDClassifier & Metamodel predictions)	0.87333

Incredibly, this approach allowed us to increase the prediction quality on the private test up to 87.3%!
It might be worth experimenting with other coefficients, as well as checking the quality without the metamodel predictions (most likely, they are redundant), but there is no such a possibility anymore. Since the window during which it is possible to make the final submission has already closed, this result will be the final one.

Final accuracy obtained on the test - 87.3%

Final solution:

MEAN(
3xBERT_predicted_logits &
BestSGDClassifier_decision_function_value &
Metamodel_raw_predictions
)