CS412 Term Project

Ufuk Ulaş Tokat 28914
Hüseyin Berke Fırat 29011
İbrahim Mahir Akbaş 29492
Ömer Can Öztürk 29248

1. Introduction **2**
2. Problem Description **2**
3. Methods **3**
3.1 Naive Bayesian 3
3.2 Logistic Regression 4
3.3 Random Forest 4
3.4 Neural Networks (MLP) 5
4. Results and Discussion **6**
4.1 Naive Bayesian 6
4.2 Logistic Regression 6
4.3 Random Forest 6
4.4 Neural Network (MLP) 7
5. Appendix **8**
5.1 Contributions 10

1. Introduction

In this homework we were given big data which is some bug definitions along with their severity classifications. Our task was training a model using them and making estimations over a test data which we don't know their real classification. We tried different learning models together with different vectorization algorithms and hyperparameters. Approaches we used to find the best model are discussed under the discussion part.

2. Problem Description

We were given a training dataset structured as shown in Figure 1. The "Summary" column contains the text data used for training the algorithm (X), while the "Severity" column holds the actual classifications of the training data (y).

	bug_id	summary	severity
5941	537143	support CSS transitions on and inside 'display...	normal
123871	948654	flex items not wrapping in multi-line flexbox ...	normal
18881	525293	NJ merge: possibly the last patch	normal
56111	1035993	spdy sessions should be threadsafe as nshttpch...	normal
143453	723975	intermittent crash (out of memory?) in crashte...	normal

The first challenge in this task is converting the textual data into a numerical format that can be used by the model. As discussed in the methods section of this report, we employed two techniques for this purpose: CountVectorization and TfidfVectorizer.

The second challenge involves selecting an appropriate model for training. Our dataset consists of seven severity classes for bugs: trivial, enhancement, minor, normal, major, blocker, and critical.

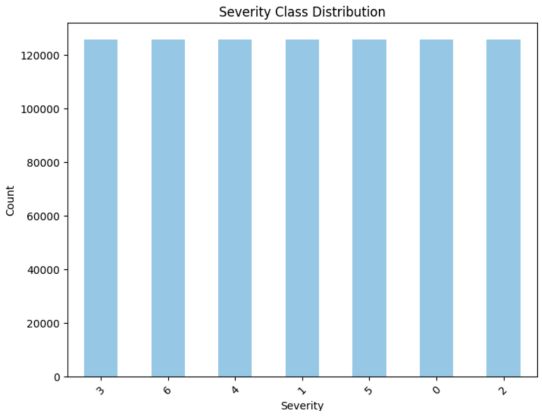
A third challenge arises from the imbalance in the data. Specifically, the "normal" class contains significantly more samples compared to the other classes.

3. Methods

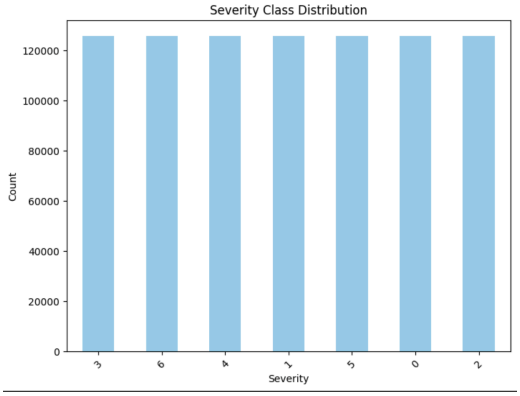
3.1 Naive Bayesian

The first algorithm we tried was naive bayes. It is a naive algorithm that assumes independence of the features as it is in the name. We used the gridSearch method of the Sklearn library to look for different hyperparameter combinations for both naive bayes and vectorizer algorithms. Our vectorizer was CountVectorizer and the classifier was MultinomialNB from Sklearn.

We attempted to handle the imbalance of the dataset by using the methods we learned in class. First, we used the SMOTE sampling method of "*imblearn" library* to match the number of datapoints in the minority class with the 'normal' labeled majority class. The second attempt was reducing the majority first with RandomUnderSampling method of "*sklearn"* library and then oversampling the minority class with SMOTE again. Since the dataset was too large and the imbalance was too deep, we couldn't achieve any improvements with sampling techniques. Here's the diagrams displaying the distribution of classes with respect to their specified labels (0-6, where '0' denotes trivial class).



SMOTE only



RandomUnderSampling and SMOTE

We tried following values for the hyper parameters without using any imbalance methods:

```
'vectorizer__ngram_range': [(1, 1), (1, 2)],  # Uni-gram or bi-gram
'vectorizer__min_df': [1, 2, 3],  # Minimum document frequency
'vectorizer__max_df': [0.9, 0.95, 1.0],  # Maximum document frequency
'classifier__alpha': [0.1, 0.5, 1.0],  # Smoothing parameter
```

In this part we preprocessed the data for only separating english words and punctuations. We used 5 fold cross validation to compare performance of the models in grid search. Then we trained the model with the best accuracy on the data and applied to test data for predictions.

3.2 Logistic Regression

Second algorithm we tried was logistic regression. It was a more statistical approach to the problem. We used the same gridSearch algorithm with 5 fold cross validation for logistic regression. Our parameter grid was as followed:

```
'vectorizer__ngram_range': [(1, 1), (1, 2), (2, 2)],
'vectorizer__min_df': [1, 2, 3, 5],  # Minimum document frequency
'vectorizer__max_df': [0.5, 0.7, 0.9],  # Maximum document frequency
'vectorizer__stop_words': ['english'],  # Including stop words removal
'vectorizer__max_features': [1000, 5000],  # Maximum number of features
'classifier__C': [0.1],  # Inverse of regularization strength
'classifier__penalty': ['l1'],  # Regularization type
```

We used a liblinear solver for the logistic regression models we tested. Before the grid search evaluation we set the max number of iterations to 500 for gridsearch to take less time. After the evaluation we increased the max number of iterations of the best model to 5000 for the model to converge best weight values.

Then we work on the class weights to handle the imbalance in the data. We decreased the weight of the normal class because it was way too oversampled. Then we tried to increase the weights of the minority classes. We even tried randomly undersampling normal classes.

3.3 Random Forest

The third algorithm we used was random forest. It is an ensemble approach to the problem. We did not use gridSearch this time because random forest can take too much time when some of its hyperparameters go higher. We manually shuffled and split the training bugs data into train split and test split. Then trained the model one by one

on the train split in a for loop. Then we evaluated each of them on the test split. The hyperparameters we tried are listed below:

```
'n_estimators': [100, 200],
'max_features': ['sqrt', 'log2'],
‘max_depth': [20, 30, None],
'min_samples_split': [5, 10],
'min_samples_leaf': [2, 4]}
```

We did not try different vectorizer parameters because we already chose the best vectorization parameters in previous models. But we used TfidfVectorizer this time.
To handle the imbalance data we tried a balanced random forest model as well.

3.4 Neural Networks (MLP)

The fourth algorithm we used to train our data and create a classifier was to use the aid of neural networks. Just like with the Random Forest algorithm, after a few tries, we found that using RandomSearch for finding the best parameters would be more feasible than a GridSearch as mlp takes a long time to process.

We created a TextProcessor class this time to preprocess the data. Then, we extracted our training and test data using the pandas library.

After that, we have created a vectorizer for our pipeline. Then, we have defined our MLPClassifier with a maximum iteration count of 500. Optimally, the number could need to be much higher, as the model may not converge even after 500 iterations at times. We have also set up an array of different possible parameters to find the best ones that work for our dataset.

Finally, after setting up our classifiers, vectorizers and possible parameters, we start the RandomSearch cross validation with feasible parameters to find which ones are the optimal parameters. We do this by then fitting the result to our training data. The yielded result is the best_model should be in theory, the model that gives the best accuracy score. Finally, we fit the test data using our returned optimal model, predict the severity labels, and save them in a .csv file named “mlp_preds.csv”.

4. Results and Discussion

4.1 Naive Bayesian

We tried 54 different hyperparameter sets for the naive bayes model. Best model of them was the one with following hyperparameters:

1. Alpha = 2.0
2. Max_df = 0.7
3. Min_df = 2
4. Ngram_range = (1,2)

Its cross validation accuracy score was 0.8531919128332135. When we applied the model on the test data we were given, we had an accuracy of 0.47232 in competition. Results of other hyperparameters can be seen in appendix 5.1.

4.2 Logistic Regression

We tried 72 different hyperparameter sets for the logistic regression model. Best model of them was the one with following hyperparameters:

1. C = 0.1
2. Penalty = L1
3. Solver = Liblinear
4. Max number of iterations = 5000

Its cross validation accuracy score was 0.8549731871648396 When we applied the model on the test data we were given, we had an accuracy of 0.62926 in competition. Results of other hyperparameters can be seen in appendix 5.2.

4.3 Random Forest

We tried 32 different hyperparameter sets for the random forest model. Best model of them was the one with following hyperparameters:

1. max_depth = None
2. max_features = 'log2'
3. min_samples_leaf=1

4. n_estimators=135

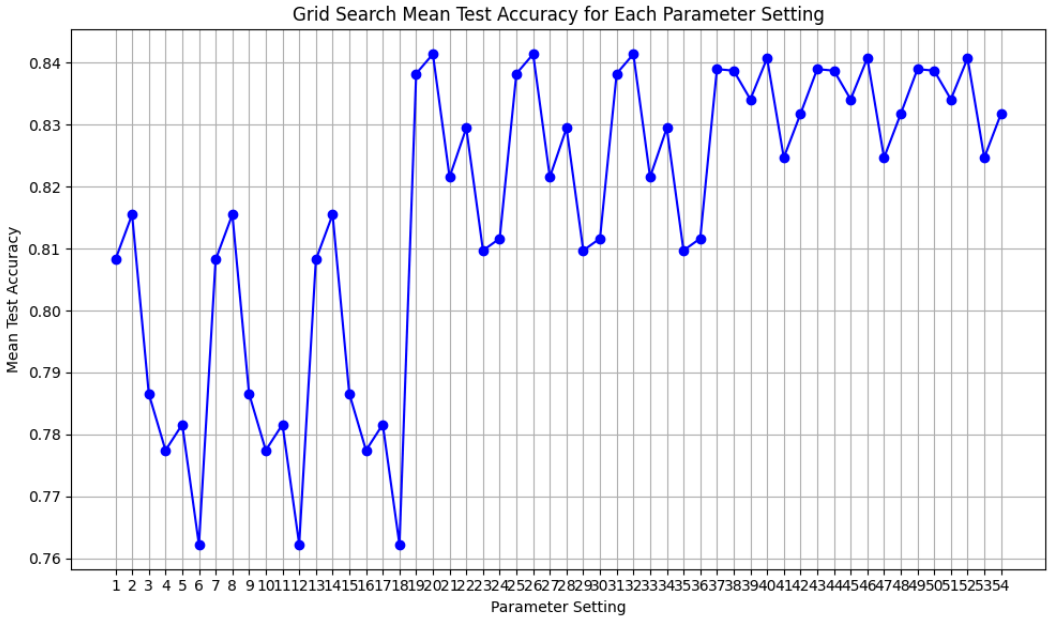
Its accuracy score was 0.85890625 When we applied the model on the test data we were given, we had an accuracy of 0.60798 in competition. Results of other hyperparameters can be seen in appendix 5.3.

4.4 Neural Network (MLP)

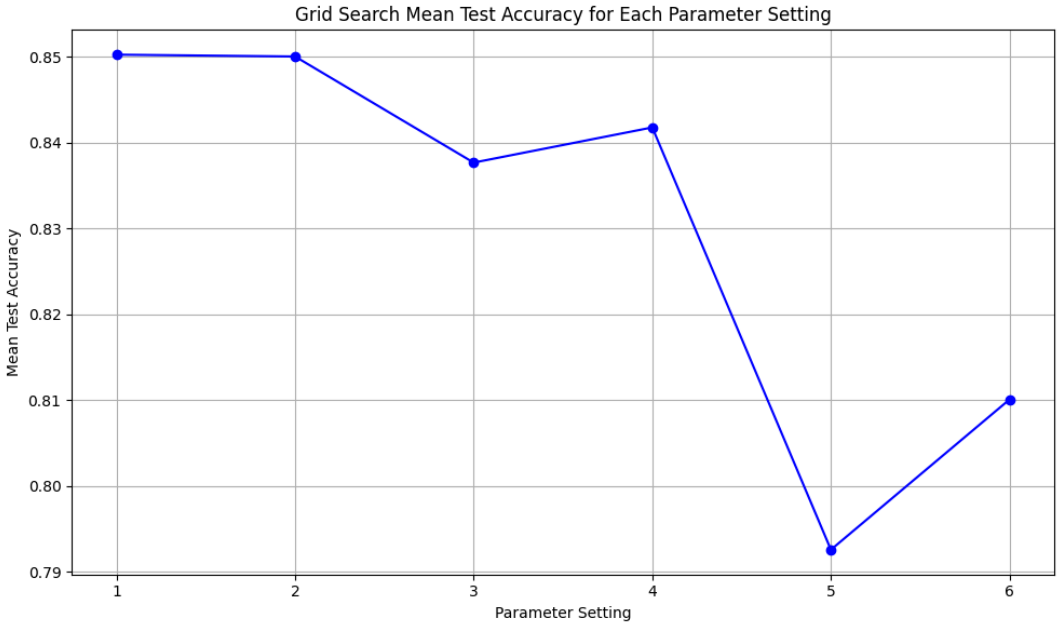
Unfortunately, due to the nature of our large parameters and MLPClassifier's exhaustive nature, we were unable to obtain an accuracy score if we try to find the optimal parameters. However, with predefined parameters the classifier returns an accuracy score, albeit with below average performance and correctness according to the Kaggle competition, scoring approximately 0.3. You can find the resulting loss rates and macro precision scores when we use the classifier with user-selected sub-optimal parameters in the appendix section figure 5.4

5. Appendix

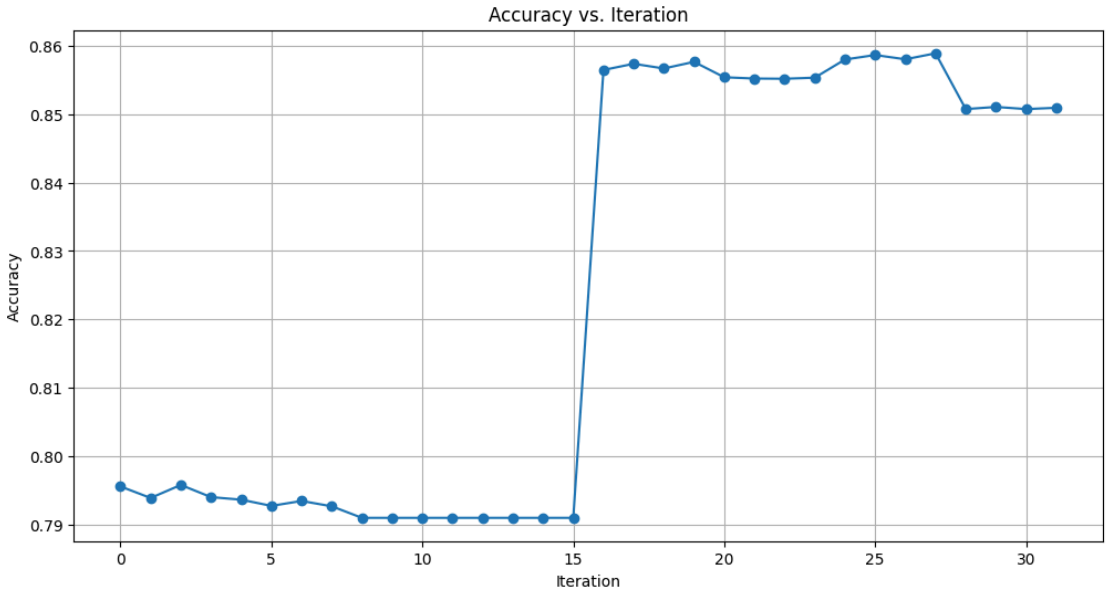
Ap.1. Accuracy of naive bayes algorithms we tried with respect to parameters we used:



Ap.2. Accuracy of logistic regression algorithms we tried with respect to parameters we used:



Ap.3. Accuracy of random forest algorithms we tried with respect to parameters we used:



Ap.4. Resulting predictions and macro precision score with the MLPClassifier without parameter selection (max_iter = 20 for demonstration purposes):

```
Iteration 1, loss = 0.68564204
Iteration 2, loss = 0.52701336
Iteration 3, loss = 0.49888376
Iteration 4, loss = 0.48314931
Iteration 5, loss = 0.47126667
Iteration 6, loss = 0.46190306
Iteration 7, loss = 0.45319627
Iteration 8, loss = 0.44548409
Iteration 9, loss = 0.43728145
Iteration 10, loss = 0.42903375
Iteration 11, loss = 0.42103753
Iteration 12, loss = 0.41254254
Iteration 13, loss = 0.40384662
Iteration 14, loss = 0.39497306
Iteration 15, loss = 0.38628948
Iteration 16, loss = 0.37820409
Iteration 17, loss = 0.36954142
Iteration 18, loss = 0.36094725
Iteration 19, loss = 0.35243518
Iteration 20, loss = 0.34423387
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (20) reached and the optimization hasn't con
  warnings.warn(
Validation macro precision: 0.8585665496016753
Test data with predictions:
       bug_id                                            summary  \
0      1143402  Firefox claims to be not the default browser w...
1      1143405  Background of html and body element are not ap...
2      1143409  Mouse input breaks after using window.showModa...
3      1143411  Build failure with next freetype version/curre...
4      1143417  HTML element is not treated as root inside for...
...        ...                                                ...
86089  1426166      Crash in bool IsAboutToBeFinalizedInternal<T>
86090  1426171  Potential crash if GraphRate is greater than 4...
86091  1426173  Crash in <name omitted> | decltype JS::Dispatc...
86092  1426174  Crash in xul.dll@0x28145fa | xul.dll@0x3c748ff...
86093  1426176  No symbols for clang_rt.asan_dynamic-x86_64.dl...

      predicted_severity
0                 normal
1                 normal
2                 normal
3                 normal
4                 normal
...                  ...
86089           critical
86090           critical
86091           critical
86092           critical
86093             normal
```

5.1 Contributions

Ufuk Ulaş Tokat: Trained logistic regression models and evaluated them to find the best logistic regression hyperparameters for the problem. Also worked on vectorization parameters to turn text data into statistical data. Then ı achieved the best parameters for the vectorization. Then we used them on other modelsç

İbrahim Mahir Akbaş: Focused on how to preprocess dataset. Tried different imbalanced dataset handling approaches. Trained RandomForest classifier models with different hyperparameters to achieve desired accuracy levels.

Ömer Can Öztürk: Worked on neural network classification models with MLP and tried to train the data on a pipeline built around it. Also conducted random search cross-validation to find optimal parameters for training our data. Calculated accuracy scores without optimum parameter selection. In the end, tried to improve the logistic regression classifiers with better parameterization and by using more accurate classifiers in the ensemble classifier.

Hüseyin Berke Fırat: Analyzed and manipulated the data with different combinations of imbalance handling methods such as SMOTE and random undersampling, however, couldn't get any meaningful results. Trained a Naive Bayesian model assuming each explanatory variable is independent. Tested different combinations of hyperparameters using grid search method.