UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Applied Machine Learning

PA3: Sentiment classification

Group 27:

Tim Boleslawsky, 980524-6478, Applied Data Science Master's, gusbolesti@student.gu.se
Mikołaj Nowacki, 20010918-2014, Applied Data Science Master's, gusnowami@student.gu.se

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part of other solutions.

1. Business Understanding

The primary objective of this project is to develop a text classification system for Birch that is performant, explainable, and maintainable. The system should not only achieve strong predictive performance but must also be interpretable and robust enough to support long-term usage within the organization.

There are several key considerations guiding the development of this system:

- **Users:** The system will be used by both technical and non-technical colleagues at Birch. Therefore, it must produce results that are accessible and understandable to non-experts, while also providing technical depth for further analysis by data scientists and engineers.
- **Deployment:** The system is intended to be deployed in a production environment where reliability and consistency of performance are crucial. This includes handling new and unseen data with minimal degradation in quality.
- **Success Criteria:** The success of the system will be measured by its accuracy in classifying text, the interpretability of its decisions, and its robustness to noisy or unexpected inputs.

To achieve these goals, we are provided with three datasets: **Crowd Dataset:** A large, crowd-sourced dataset annotated by non-experts. **Gold Dataset:** Same data points as crowd-sourced dataset, but higher-quality and trusted labels. **Test Dataset:** Reserved for the final evaluation of the system's performance. Contains trusted labels and different tweets than the training sets.

2. Data Munging and Exploratory Data Analysis

Before we develop any models, we need to make sure the data is appropriate for machine learning tasks. In this section, we want to focus on making sure the data has no missing values, no obvious errors, and that the data within and between the different data sets is comparable.

To look for missing values we check the sentiment column of each data set for null values and sum the number of null occurrences. If this sum is 0 we can be sure there are no missing values. For our data sets we find no missing values and can continue.

Next we need to make sure all the labels are compatible and comparable and that there are no errors in the labels. To validate this, we check the unique values in the sentiment column of the data sets. Taking a look at the result of this analysis, we can see that for the gold and the test data set, we have the appropriate labels, namely: ”neutral”, ”positive”, and ”negative”. For the crowd-sourced data set, we see a lot of errors:

- “Positive”, “Negative”, “Neutral”, “neutral”, “negative”, “positive”, “neutral?”, “Nuetral”, “negative “, “positive “, “neutral “, “ negative”, “Nedative”, “:_x0008_neutral”, “netural”, “netutral”, “Neutral “, “Neural”, “Neutrall”, “Netural”, “neugral”, “neutrla”, “negtaive”, “postive”, “ neutral”, “ positive”, “ positive”, “ neutral”, “ negative”, “Nutral”, “negayive”, “positie”, “neutra l”, “npositive”, “neutal”, “postitive”, “Positve”.

To ensure that only appropriate labels remain, we use string functions and fuzzy matching to map each of the faulty labels to the corresponding correct one. First, we normalize the string to lower case and remove punctuation. Then a fuzzy matcher tries to match each label in the crowd-sourced data set to the predefined labels (”neutral”, ”positive”, ”negative”). After applying this cleaning function we get the appropriate and expected labels for the crowd-sourced data set.
For the next step in this section we want to check if the training data we have is reliable. In our case this means checking if the annotators of the data performed their work well. To check this we look at the consensus between the crowd-sourced data set and the gold data set. A common metric to check the inter-annotator agreement is the Cohen kappa score. If we look at this score between the crowd-sourced data set and the gold data set, we get a value of about 0.193. This unfortunately means that the crowd-sourced data set and the gold data set do not agree often, which leads us to the conclusion that the crowd-sourced data set is of poorer than desired quality and maybe should be re-annotated with clearer instructions.
Doing exploratory data analysis in our case consists of analyzing the sentiment distribution in the unique data sets. To visualize this, we create three bar charts to show the number of labels for ”positive”, ”negative”, and ”neutral” sentiment in each data set. This can be seen in figure (1).

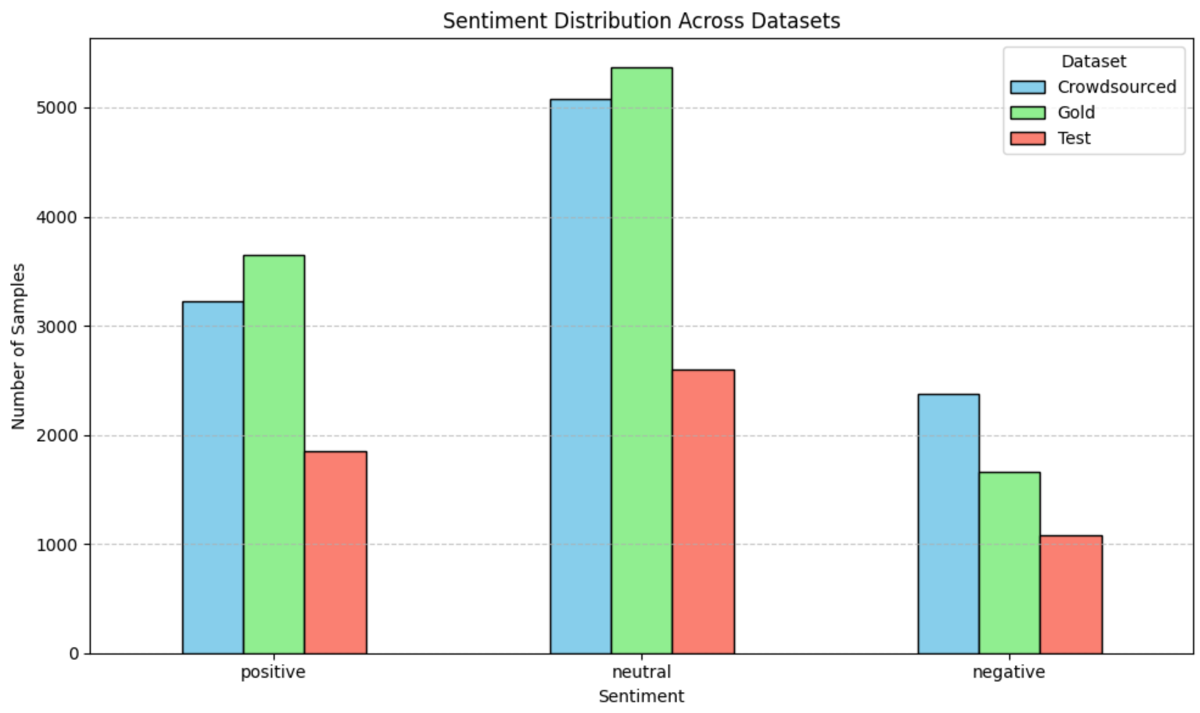

Figure 1: Data distribution

3. Data Preprocessing

An important step of data preprocessing is data encoding. By parsing textual data into numerical form, we enable the use of machine learning models, which typically require numerical input. Because we just want to convert the target variable into numerical values, we use a `LabelEncoder` rather than something like One-Hot Encoding. Now, instead of $[negative, neutral, positive]$ the sentiment is represented like this: $[0, 1, 2]$.

In the next step, we divide the dataframes into single arrays: X for text and Y for encoded labels; for improved readability and to adhere to the machine learning standards.

In the last step of this section, we apply `TfidfVectorizer` which numerically represents the importance of each word in the data. Because our feature is text, we need the `TfidfVectorizer` to make the text data suitable for machine learning models (which again, usually require numeric input). This step is usually called feature engineering or feature extraction.

In typical machine learning projects, we perform a train-test split, but here it is unnecessary since the datasets already provide this split. Similarly, although there is a noticeable imbalance between sentiment classes (around a 1:3 ratio between negative and neutral, see Figure 1), it is within acceptable limits. Moreover, our chosen classifiers have built-in mechanisms to handle such slight imbalance.

4. Building the Model

Firstly, we created a baseline performance model to evaluate against our further improvements. In our case, we just use a simple dummy classifier that guesses the most frequent label for all data points. The dummy classifier achieved a cross-validation score of 0.48.

Secondly, we implemented 3 different classifiers to compare their accuracy and performance. We selected `LogisticRegression`, `Multinomial Naive Bayes` and `LinearSVC`. We chose these classifiers for a few reasons. First, text data represented as bag-of-words or TF-IDF often results in very high-dimensional feature spaces. All three classifiers handle sparse, high-dimensional data efficiently. Second, all three algorithms are relatively computationally efficient, if we, for example, compare them to tree-based classifiers. This makes them practical for large text corpora. Third, the coefficients from these models can be interpreted to understand word importance for each class. This will make the interpretation of the results much easier. Now there are some drawbacks to these classifiers as well, especially the `Multinomial Naive Bayes`. Namely, the `Multinomial Naive Bayes` classifier has no inherent way to balance imbalanced classes, the other two classifiers do, and `Multinomial Naive Bayes` feature independence, which might not be true.
Initially, the cross-validation scores across 5 folds are as follows:
- Logistic Regression: 0.565
- Multinomial NB: 0.507
- LinearSVC: 0.558

Furthermore, we compared the computation time of the classifiers between different subset sizes. The results are presented in Table 1: We do this to get a feel for the scalability of the classifiers. From the initial cross-validation scores, we can see that the `LogisticRegression` and the `LinearSVC` classifiers performed almost identically. But when comparing scalability, we see a huge difference in favor of the `LinearSVC`.

For that reason we choose the `LinearSVC` as the most suitable classifier for our problem.

Classifier	**10%**	**25%**	**50%**	**75%**
Logistic Regression	2.731s	4.601s	5.089s	6.515s
Multinomial Naive Bayes	0.028s	0.002s	0.003s	0.018s
Linear SVC	0.020s	0.052s	0.089s	0.133s

Table 1: Training time for different classifiers

Now we want to tune this classifier to maximize its performance by finding the best values for its hyperparameters.

The first question we need to answer is what method of hyperparameter tuning we want to use. Two common approaches are `GridSearchCV` and `RandomizedSearchCV`. For our case we choose `GridSearchCV` because our grid of parameters is small, which ones we test will be discussed later, the training time is fast, and we want an exhaustive search to guarantee we get the best model.

When tuning a `LinearSVC` we want to focus on these hyperparameters: First, we look at the hyperparameter 'C'. This parameter is important because it determines the regularization strength. Second, want to look at the 'dual' hyperparameter. Usually, we would suspect dual=True to be better with more features than samples and dual=False to be better when the inverse is the case. Lastly, we check the 'max_iter' hyperparameter, which controls the maximum iterations. After tuning the hyperparameters of Linear SVC, we can slightly improve the performance of the classifier from a cross-validation score of 0.557 to an accuracy of 57% on the crowdsourced data. The optimal values we get for the hyperparameters are: 'C'=0.5; 'dual'=False; 'max_iter'=1000.

5. Testing the Model

We performed two tests on the Linear SVC algorithm. First, we trained it on crowdsourced data and tried to predict the labels on the test set. Then, we trained another LSVC model on gold data and used it on the test set.

- For crowdsourced data we achieved 0.620 accuracy score and 0.618 F1-score on the test set.
- For gold data we achieved 0.781 accuracy score and 0.779 F1-score on the test set.

This result is interesting for a few reasons. First, we can see that no matter what data we use, we get a substantially better result than our baseline model. This gives us some confidence that the model inherently does something smart and useful. The most obvious observation one can make from this is the huge gap between the model trained on the crowdsourced data and the model trained on the gold data. This, together with the poor Cohen kappa score, leads us to the conclusion that the quality of the crowdsourced data is quite poor. This is both represented in the accuracy and the F1-score. To get a better understanding of why this happens, we want to look at the feature importance and a few mistakes of the model trained on the crowdsourced data.

Looking at the feature importance, we see pretty sensible decisions from the classifier. For the negative sentiment, for example, we have words like "worst", "sad", "fuck", "hate",

... that indicate a negative sentiment. For the positive sentiment, we get words like "happy", "love", "amazing", ... that indicate a positive sentiment.

Where it gets interesting is when we analyze some mistakes the classifier makes. When looking through these mistakes, two things become apparent. First, some text is just wrongly or at least questionably labeled. Let's take this example: "the sun is so stupid they know how much the boys mean to everyone and they knew how hard it was for us when zayn left this is so disgusting". This was labeled as "neutral" but we think this very obviously can be labeled as negative (like our model did). Second, some texts are just very hard to classify. For example this text: "@CNN So what If the rules were spelled out back in May. Politics is all about change. Carly Fiorina deserves a place in your debate now.". We would argue that it is very hard to classify this either as positive, negative, or neutral.

To get a more nuanced evaluation of the classifier (compared to just using accuracy and f1-score) we want to look at the classification report. This classification report is depicted in Table 2:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Negative	0.54	0.54	0.54	1077
Neutral	0.62	0.71	0.66	2597
Positive	0.68	0.54	0.60	1850
Accuracy			**0.62**	5524
Macro Avg	0.61	0.60	0.60	5524
Weighted Avg	0.63	0.62	0.62	5524

Table 2: Classification report for LinearSVC model

The main insight we get from this is that, like we could have assumed, the classifier has the most trouble with negative sentiment classification. This is very likely due to the imbalance in the training data.

While the model demonstrates a clear improvement over the baseline, several aspects indicate room for enhancement:

- **Handling Imbalanced Data:** Although class imbalance is moderate, we see that it has a slight effect on the performance of the model when detecting negative sentiment. Here more data points in the training data set with negative sentiment could help.
- **Impact of Data Quality:** The poor Cohen's kappa score (approximately 0.19) observed during inter-annotator agreement analysis and the analysis of wrongly classified examples suggests significant inconsistency in the training data annotations, especially in the crowd-sourced dataset. This low agreement undermines the model's ability to learn reliable patterns and may be a major reason for the modest classification performance, especially when compared to the gold data set. Re-annotating the crowd-sourced data with stricter guidelines and quality control could substantially boost model accuracy and generalizability.