Exploring Classifiers for Automated Review Topic Detection

Chris van Riemsdijk & Herman Lassche & Shrushti Kaul
s3997049 & s4108353 & s5288843

Abstract

This research explores classifiers for automated review topic detection. We use a multitude of pre-processing methods, feature vector creations, and classifiers to find the best model to automatically detect topics on 6,000 online Amazon reviews. For this research, we created a pipeline to train, evaluate, and test classifiers and employ different hyperparameters. By using this pipeline, we discovered by using a linear SVM kernel that we are able to correctly identify 93% of new unseen samples with a macro F1 score of `0.93` over all 6 categories.

1 Introduction

In the 21st century, people communicate more than ever. This communication often goes by writing. The amount of written text keeps growing, and thus the amount of data. Automatic classifiers can help to structure this data. This research paper will take reviews into account. You can get reviews of any restaurant, product, or even person you wish to learn more about by conducting a quick Google search. However, when a person is just interested in a certain type of review, it may be difficult to sift through all the reviews until the relevant ones are discovered. Therefore, this research aims at structuring review data. The data should be divided into six different product categories.

Reviews are characterized by the shortness of the message. Sometimes a review consists of only a few words or a single sentence. In the NLP field of study is categorization a common theme. However, many papers focus on larger documents or online posts. These corpora have different characteristics from the review dataset, therefore the classification algorithms could need other tweaking and design choices.
Our research question is as follows:

RQ: Which (combination of) feature vectors and classifiers generalizes the best on a corpus of reviews obtained from Amazon?

We will experiment with classifiers such as Naive Bayes, Decision Trees, Random Forests, K-Nearest neighbours, and Support Vector Machines. We will also dive deeper into the multitude of feature vectors. Examples of those are part-of-speech tags, n-gram sizes, removing correlated features, and combining features with each other.

2 Related Work

2.1 Classifiers

A classifier is a machine learning model that has been trained to classify a dataset into predetermined categories. A classifier's main task is to allocate a probable class to all input data points using patterns and characteristics acquired from a training dataset.

This research considers the following methods of classification. Methods to consider are *Naive Bayes*, *Decision Trees*, *Random Forests*, *K-Nearest Neighbours*, and *Support Vector Machines*.

Naive Bayes is a supervised algorithm based on calculating probabilities via Bayes' formula; it is a simple, yet very widely used framework for classification (Lewis, 1998). The Bayes' formula can be represented as the following

$$P(A|B) = \frac{P(BA) * P(A)}{P(B)} \tag{1}$$

wherein, $P(A|B)$ is the posterior probability of event A occurring given that event B has occurred,

$P(B|A)$ is the likelihood of event B occurring given that event A has occurred, $P(A)$ is the prior probability of event A occurring, $P(B)$ is the probability of event B occurring.

Within the context of classification, a class from a set of classes $(c_1, c_2, \ldots, c_c)$ is to be identified for a set of attributes. Denoting a label for a class as C, and a set of n features $(X_1, X_2, \ldots, X_n)$, the Bayes' formula can now be treated as a classification problem i.e. probability of class C, given the values of features X_1 to X_n:

$$P(C|X_1..X_n) = \frac{P(X_1, X_,..X_n|C) * P(C)}{P(X_1, X_,..X_n)} \tag{2}$$

The "naive" part of Naive Bayes comes from the assumption that the features are conditionally independent given the class, which makes the term $P(X_1, X_2, ..X_n|C)$ easier to deal with. Although feature independence is generally a poor assumption, in practice Naive Bayes often competes well with more sophisticated classifiers (Rish, 2001).

Support Vector Machines The idea of Support Vector Machines was introduced back by (Vapnik, 1963) as the Generalized Portrait algorithm. (Cortes and Vapnik, 2009) explains that an SVM conceptually implements the following idea: input vectors are non-linearly mapped to a very high-dimensional feature space. In this feature space, a linear decision surface is constructed. Special properties of the decision surface ensure the high generalization ability of the learning machine. The decision hyperplane is determined by a weight vector, w and a bias term, b, through a decision function

$$f(x) = w * x + b \tag{3}$$

SVM aims to find the hyperplane that maximizes the margin between the two classes while minimizing the classification error. The margin is calculated as the distance between the hyperplane and the closest data point. Mathematically,

$$M = 2/||w|| \tag{4}$$

where $||w||$ is the magnitude of the weight vector. Thus, the optimal hyperplane is derived by trying to minimize $\frac{||w||^2}{2}$ while ensuring that all data points are correctly classified. The data points from the training dataset that are closest to the decision boundary are called support vectors. These are the data points that have the smallest margin or the points that are most challenging to classify correctly. Support vectors are crucial in SVMs because they determine the position and orientation of the decision boundary. The separation of data points relies on the choice of a kernel function, which comes in various types such as *linear*, *radial basis function*, *sigmoid*, and so on. In essence, these functions govern how well and smoothly the classes are separated, and adjusting their hyperparameters should be done carefully to avoid overfitting or underfitting issues. However, SVM inherently only performs binary classification. For multi-class classification, one of the methods needs to be employed

- One-vs-All strategy: a binary classifier is created for every single class, wherein a classifier treats one class as the positive class, and the rest as negative. This is the better strategy to choose when there are a large number of classes to classify instances against.

- One-vs-One strategy: a binary classifier is created for every possible pair of classes. Every classifier is trained using a subset of the dataset that includes only the instances corresponding to the two classes it is intended to differentiate. OVO is more robust than its former alternative.

Decision Trees A decision tree serves as a classifier represented as a recursive division of the instance space. This tree comprises nodes that collectively form a rooted structure, which means it's a directed tree featuring a node designated as the "root" with no incoming connections (Rokach and Maimon, 2005). All other nodes possess precisely one incoming connection. A node that sends out connections is referred to as an internal or test node, while all remaining nodes are termed leaves, also known as terminal or decision nodes. Within a decision tree, each internal node subdivides the instance space into two or more sub-spaces based on a specific discrete function of the input attribute values. In the most basic and common scenario, each test pertains to a single attribute, thereby segmenting the instance space according to the attribute's value. In instances involving numeric attributes, the condition pertains to a specific range of values. (Rokach and Maimon, 2005) also states top-down decision trees have a clear preference in the literature, some of the most prominent inducers of them being:

- Iterative Dichotomiser: ID3(Quinlan, 1986) is an early decision tree algorithm that uses entropy and information gain to build classification trees.
- C4.5(Quinlan, 1993): It is an improvement on ID3; it can handle both categorical and continuous data, using information gain and gain ratio for attribute selection.
- Classification and Regression Trees: CART (Breiman et al., 1984) is a versatile decision tree algorithm that can be used for classification tasks, employing Gini impurity or mean squared error for node splitting.

Typically, in the majority of cases, the discrete splitting functions focus on individual attributes, meaning that an internal node is divided based on the value of a single attribute. As a result, the algorithm seeks to identify the most suitable attribute for the split. Some of the most common splitting criteria in the literature are impurity-based (Rokach and Maimon, 2005), some listed as follows

- Information Gain: It is an impurity-based criterion that uses the entropy measure (origin from information theory) as the impurity measure (Quinlan, 1987).
- Gini Index: Gini index is an impurity-based criterion that measures the divergences between the probability distributions of the target attribute's values. (Rokach and Maimon, 2005)

Random Forests One way that decision trees can maintain their accuracy is by forming an ensemble via a random forest algorithm (Breiman, 2001). Random forests are an ensemble of tree predictors where each tree's decisions are influenced by a random vector, independently sampled and following the same distribution for all trees within the ensemble. Random Forests leverage randomness through bootstrapping and random feature selection to create a diverse ensemble of decision trees. This diversity and the averaging or voting of predictions lead to improved generalization performance. At each node of a decision tree, a random subset of features (columns) from the original feature set is considered for the split, introducing this helps reduce the correlation among trees, making the ensemble more robust.

As the number of trees in the forest grows, the generalization error of the ensemble converges to a limit. The ensemble's generalization error is influenced by both the individual tree's predictive power and the degree of correlation among them.

K-Nearest Neighbours (KNN) Initially introduced by (Fix and Hodges, 1985), this algorithm is a supervised learning classifier that relies on closeness to classify or forecast the category of an individual data point. It's generally employed for classification tasks, based on the idea that comparable data points tend to be located in proximity to each other. KNN essentially makes use of "majority voting", wherein occurrences of each class are counted among the K nearest neighbours. The class that appears most frequently is chosen as the predicted class for the data point in question. There are various distance metrics that can be used in KNNs, like *Euclidean distance*, *Minkowski distance*, and so on. The choice of the distance metric and the value of K are important hyperparameters that can significantly affect the performance of a KNN model. However, KNN becomes increasingly inefficient as the input dataset increases, compromising the overall model performance.

3 Method

3.1 Data

The used data in this research comes from various English reviews. The reviews can be split into six categories: `books`, `camera`, `dvd`, `health`, `music`, `and software`. The reviews are gained from Amazon[1] concerning products from one of these six topics. The corpus consists of *6000* reviews in total. The set is divided into a train set, a dev set, and a test set. Respectively of the following size: *4800*, *600*, and *600*. Following a $80\% - 10\% - 10\%$ split.

Figure 1 and Table 1 show the data distribution among the six different topics. Only the training set is used to visualize the distribution since the model will be tuned based on this distribution. As can be seen, the data is well-balanced, therefore we do not see a need to use techniques to counteract imbalance in the data.

The used dataset is already pre-processed (e.g. lowercase etc.) No additional pre-processing was applied. Though, pre-processing methods are used

[1] https://amazon.com

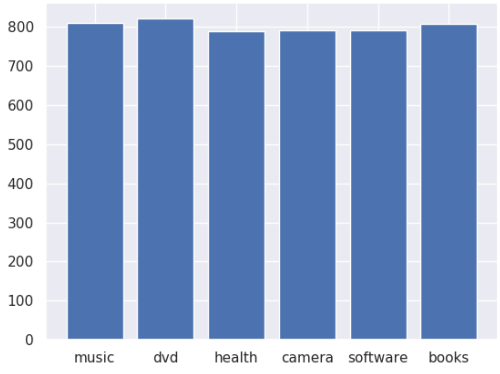

Figure 1: Distribution of the reviews among the topics (training set)

Topic	Count
books	807
camera	790
dvd	821
health	789
music	809
software	790
Total	6000

Table 1: Distribution of the reviews among the topics (training set)

during the experiments, to see how they affect the feature sets. Those additional pre-processing methods are discussed in the relevant sections.

Furthermore, the data is annotated with a binary classification of the sentiment of the review. A review is considered either to have a positive or a negative sentiment. This label is not used in this research.

3.2 Experiments

Different n-gram sizes As a feature vector Bag-of-words (BOW) and TF-IDF are considered. For these methods, the number of words to take into account is a hyperparameter, the so-called `n-gram`. This parameter should be tuned. The considered n-grams are: `[1], [2], [3], [1,2], [1,2,3], [2, 3]`. Using a larger range (>3) would lead to large entities, which would cause a low probability that this exact combination will exist in another review. Therefore, a larger range would probably lead to worse results. As an example, consider the following sentence: *This is an example*. Respectively the following words would be considered based on the n-gram range: [*this, is, an, example*], [*this is, is an, an example*], [*this is an, is an example*], [*this, is, an, example*, *this is, is an, an example*] etc.

The BOW and TF-IDF are implemented using the Scikit package (Pedregosa et al., 2011). The provided methods offer the ability to define an n-gram range. This ability is used to conduct the experiments per n-gram range. The tests are conducted on all five models using the best-performing configuration. In the case of KNN, the configuration is $k = 8$ and with the weighted property. The SVM configuration with a linear kernel and $C = 1$. The decision tree uses Gini and the random property. The random forest implementation also uses Gini.

Combining feature vectors As discussed before, we use BOW and TF-IDF to generate feature vectors. This works, but we can also concatenate the features. Concatenating the feature vectors gives a bigger feature space for the classifier to fit on. This might result in better generalizability for a classifier, but it can also have a negative effect. Therefore, it is important to run benchmarks on your validation set to see whether it has a positive impact. We can use Scikit's learn `FeatureUnion` to combine the features (Pedregosa et al., 2011).

Lemmatizing & Stemming Words take many forms, however, they mostly have the same meaning. An example is the verb "to cook", which can appear as "cook", "cooked", "cooks", "cooking" etc. The base form however is "cook" and this is something you would see in a dictionary. This form is called a `lemma`. A lemmatizer takes as input a non-lemma and transforms this into its lemma form. We do not do this only with verbs, but also with other words that take an inflected form. Stemming is used to reduce inflected words to their most basic form. An example of stemming is suffix-stripping, where words ending in -ing, -ed, and -ly will be condensed to their more basic form without their suffixes.

Word category count Besides BOW and TF-IDF, other methods exist to extract features from textual data. One of these methods is word counting. In this research, Empath is used (Fast, 2016). Empath counts how many words of pre-defined categories exist in the text. These (normalized) counts can be used as a feature for classification

algorithms[2]. As an example, when a review has many words from the categories 'crime', 'school', 'journalism', and 'superhero'; the review is more likely from the topic `books` than from the topic `Software`.

Removing correlated features A feature set with highly correlated features could have a negative impact on the performance of a classification algorithm (Chen-An Tsai, 2023). In this research, the feature set of the word count will be used to test the impact of removing highly correlated features. When two features are highly correlated, one of the two will be removed. Due to technical limits, it is not possible to remove the correlated features for BOW and TD-IDF. Correlation is calculated with pandas correlation function[3]. A feature is considered highly correlated if the correlation is larger than 0.85.

Part-of-speech tags In linguistics, we categorize words under certain labels. We have nouns, verbs, adjectives, adverbs, etc. These are called part-of-speech (POS). These POS have acronyms that correspond to the particular POS, so-called, tags. Therefore, we can use the tags and add them to the feature space. We hope to observe that the model can generalize better to both the text and its corresponding POS tags.

Classifiers After generating the features, we will feed the generated features to a classifier. As described in the related works, we focused on the following classifiers: *Naive Bayes*, *Decision Trees*, *Random Forests*, *K-Nearest Neighbours*, and *Support Vector Machines*. Merely using these out of the box can result in good generalization, but generally, you would need to fine-tune the hyperparameters of each classifier to yield better generalization and results. This was done by using combinations of all aforementioned experiment types and changing the hyperparameters of the model itself. From this, we can see which classifier and feature vectors are the best for the dataset. Following are the results on the `development` split. For each experiment, we used the best base-performing models to see which feature vectors would be the best.

[2]https://github.com/Ejhfast/empath-client
[3]https://pandas.pydata.org/

3.3 Different n-gram sizes

Figures 2, 3, 4, 5, and 6 show the results for all tested n-grams. In general, it can be observed that the larger the n-gram, the worse the result. Comparing the results of n-grams with size 1 and with size 3 shows that size 1 has better results in each case. The results are better when n-grams of sizes 1, 2, and 3 are combined than when n-grams of size 3 are used alone. However, it shows constantly better results when using a single n-gram of size 1 rather than several n-grams.

The observation that a smaller n-gram has better results than a large n-gram could be caused by the fact that reviews are often short texts. Therefore, combinations of more than one word do not appear very often in multiple reviews. A large n-gram would lead to a lot of unique word combinations. If a review has word combinations that do not exist in other reviews, the classification would be a guess.

In the case of reviews would be advised to use an n-gram of size 1.

Figure 2: Heatmap of Naive Bayes for different n-gram ranges — trained on train set, tested on dev set

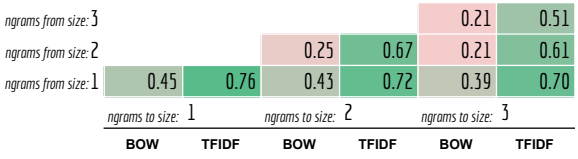

Figure 3: Heatmap of KNN (Weighed, k=8) for different n-gram ranges — trained on train set, tested on dev set

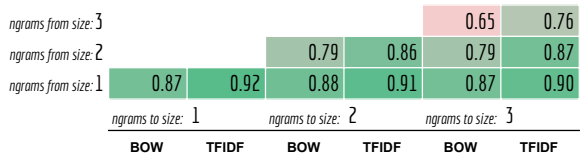

Figure 4: Heatmap of SVM (Linear, C=1) for different n-gram ranges — trained on train set, tested on dev set

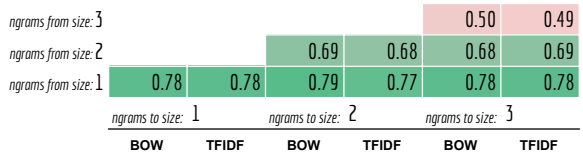

Figure 5: Heatmap of Decision Tree (Gini, random) for different n-gram ranges — trained on train set, tested on dev set

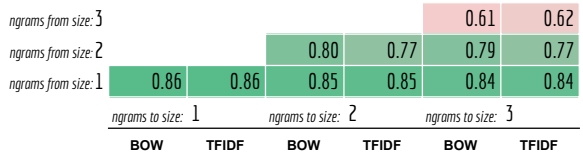

Figure 6: Heatmap of Random Forest (Gini) for different n-gram ranges — trained on train set, tested on dev set

3.4 Word category count

The results of the experiment can be found in table 2. As a first observation, it can be concluded that it is possible to predict the classes based on word count. A random guess would lead to a score around 0.17, the results are much better. The SVM is best performing, with reasonable accuracy but also for each different topic somewhat equal scores. Different from Naive Bayes, where `books` have an F1 of 0.85 and `camera` only 0.57. Overall classifiers do, some categories have much worse results than other categories. It could be possible that Empath has more relevant categories for `books` than for `camera`. Or that `camera` has more overlapping categories with another topic, which will cause the classifier to predict the other class.

Classifier	Accuracy	books	camera	dvd	health	music	software
Naive Bayes	0.71	0.85	0.57	0.58	0.64	0.84	0.71
KNN[a]	0.71	0.83	0.57	0.63	0.57	0.87	0.76
SVM[b]	0.78	0.88	0.68	0.72	0.73	0.87	0.76
Decision Tree[c]	0.64	0.74	0.55	0.5	0.60	0.82	0.62
Random Forest[d]	0.76	0.86	0.63	0.69	0.72	0.87	0.75

[a] k=8; Weigthed
[b] SVC; C=1
[c] Gini; Random
[d] Gini

Table 2: Results of using the word category count as a feature vector, overall accuracy and per category F1 score — trained on the train set, tested on dev set

3.5 Removing correlated features

Comparing table 2 with table 3 outlines whether removing highly correlated features improves the performance. The performance stays equal or drops. The differences are not huge, but the performance is surely not better. Therefore, it can be concluded that removing the highly correlated features has no benefit in our case. The models' ability to handle correlated features or the absence of strongly correlated features could be one factor.

Classifier	Accuracy	books	camera	dvd	health	music	software
Naive Bayes	0.69	0.84	0.56	0.58	0.63	0.82	0.70
KNN[a]	0.70	0.84	0.56	0.62	0.55	0.84	0.75
SVM[b]	0.78	0.88	0.68	0.72	0.74	0.86	0.75
Decision Tree[c]	0.59	0.71	0.47	0.45	0.58	0.74	0.54
Random Forest[d]	0.75	0.83	0.64	0.68	0.70	0.86	0.71

[a] k=8; Weigthed
[b] SVC; C=1
[c] Gini; Random
[d] Gini

Table 3: Results of using the word category count after removing the highly correlated features, overall accuracy, and per category F1 score — trained on train set, tested on dev set

3.6 Lemmatizing & Stemming

As seen in table 4, the performance of the models is quite high on the dev set. Only the KNN has poor performance. This shows that stemming and lemmatizing the input vectors enhances the classifier's generalizability. Especially in the case of Naive-Bayes, the accuracy and F1 scores per category are high compared to all other experiments thus far.

Classifier	Accuracy	books	camera	dvd	health	music	software
Naive Bayes	0.89	0.92	0.87	0.86	0.86	0.93	0.89
KNN[a]	0.38	0.46	0.37	0.34	0.37	0.4	0.37
SVM[b]	0.88	0.89	0.85	0.88	0.83	0.93	0.88
Decision Tree[c]	0.78	0.81	0.77	0.84	0.67	0.89	0.70
Random Forest[d]	0.86	0.9	0.85	0.89	0.75	0.94	0.83

[a] k=8; Weigthed
[b] SVC; C=1
[c] Gini; Random
[d] Gini

Table 4: Results of using lemmatization and stemming, overall accuracy and per category F1 score — trained on train set, tested on dev set

3.7 Part-of-speech tags

As seen in table 5, POS tags show a positive improvement in accuracy and F1 scores on all the models except for the KNN model again. The best-performing model is a linear SVM with an accuracy of 90% and a macro F1 score of `0.90`. This shows that including POS tags may introduce better generalizability for the classifiers, as this extra data may be useful.

Classifier	Accuracy	books	camera	dvd	health	music	software
Naive Bayes	0.86	0.90	0.83	0.84	0.71	0.93	0.89
KNN[a]	0.34	0.28	0.43	0.34	0.35	0.32	0.32
SVM[b]	0.90	0.91	0.88	0.91	0.86	0.92	0.91
Decision Tree[c]	0.79	0.85	0.81	0.80	0.69	0.84	0.75
Random Forest[d]	0.88	0.94	0.87	0.88	0.81	0.92	0.88

[a] k=8; Weigthed
[b] SVC; C=1
[c] Gini; Random
[d] Gini

Table 5: Results of using POS tags as an extra feature, overall accuracy and per category F1 score — trained on train set, tested on dev set

4 Results

After running all the experiments on the development set, we could extract the best classifiers of each type e.g., KNN, SVM, etc. Now to test their generalizability, we left out 10% of the whole dataset to test on, which is *novel* data. We hope to see the same or better performance scores compared to the development set, as this shows that the data is not overfitted on the data. The best models per classifier are shown in table 6 with their hyperparameters.

Classifier	Accuracy	books	camera	dvd	health	music	software
SVM[a]	0.93	0.95	0.92	0.93	0.91	0.95	0.91
DT[b]	0.78	0.81	0.71	0.77	0.69	0.83	0.76
KNN[c]	0.77	0.72	0.79	0.72	0.76	0.77	0.81
RF[d]	0.91	0.93	0.91	0.93	0.86	0.93	0.90
NB[e]	0.91	0.95	0.90	0.88	0.90	0.94	0.91

[a] Linear, One-vs-All, C=0.5, TF-IDF, lemmatization and stemming, unigram
[b] Criterion=Gini, Splitting=Random, BOW + TF-IDF, unigram
[c] K=8, Weighted, TF-IDF, unigram
[d] Criterion=Gini, BOW + TF-IDF, lemmatization and stemming, unigram
[e] Default, TF-IDF, lemmatization and stemming, unigram

Table 6: Results of the best model of each classifier that we have trained and tested, overall accuracy and per category F1 score — trained on train set, tested on test set

As seen in 6, the linear SVM with One-vs-All and `C=0.5`, TF-IDF combined with lemmatizing and stemming the input and only using unigrams yields the best results on the test set. With an overall accuracy of 93%. Both the macro and weighted F1 scores are `0.93`.

5 Discussion

An accuracy of 93% and a macro and weighted F1 score of 0.93 are already great values. Looking back at our research question, which was: **Which (combination of) feature vectors and classifiers generalizes the best to a corpus of reviews obtained from Amazon?** We can now answer this that for this corpus of reviews, the linear SVM kernel with TF-IDF, lemmatization, and stemming has the best results. This shows that automated topic detection is possible for people who accept the current error rate of the classifier.

However, we would like to have a quick look into why we did not get a better-generalized model. We think that we missed this 7% due to outliers in the data. Some data samples can be really similar to 2 topics. An example of such a data sample is: “after have read the book and wait in anticip when it first came out , i wa extrem disappoint . the movi veri well could be alright on it own merit , but the book plot is veri differ and much more suspens”. Our classifier thinks this is about books, however, the ground-truth value is DVD. Although it is lemmatized and stemmed, it seems that the reviewer is saying that the film in itself was okay but that it was very different from the book which had way more suspense. This is a case where the model does not know if it is about the book or about a DVD/film. We could say that some humans might classify this as a book review as well.

5.1 Future Work

Although we have tested a plethora of hyperparameters, pre-processing methods, and classifiers, there is still future work. Following is an enumeration of ideas that can be used in future research:

- Doing random- and grid search to get a better search space of the hyperparameters.
- Test a machine learning model like `Word2Vec` for vectorizing the input
- Embeddings have shown great enhancement in models when used, therefore embedding the input might be beneficial research.

6 Answers to additional questions

6.1 What is the difference between `svm.SVC` and `svm.LinearSVC`?

The most prominent difference between these two classifiers is that LinearSVC uses OVA (One vs. All) multi-class reduction, while SVC uses OVO (One vs. One) multi-class reduction. Since the latter has more number of classifiers in case of classification being done between more than two classes, SVC would be more robust than LinearSVC.

6.2 Why should one not look at the test set?

The reason this is discouraged is that we want the model to generalize over the data it trains, and not overfit. A model is most of the time not perfect out of the box, therefore we can use a development set to test our results after finetuning hyperparameters. Finally, we can check how well our model has generalized on the data to use the test set as "novel" data that the model has never seen before, this mimics the real-world example for novel data and hopefully really shows if you have a good model.

6.3 Why is it useful to look at the confusion matrix?

It is a concise summary of the performance of a model built for classification tasks. It is a visual representation of how the model classifies certain classes and gives a quick overview of which classes could be misinterpreted by the model. Moreover, it enables us to calculate different performance metrics, which gives an individual a clear idea of the model's performance.

6.4 What is the difference between macro F-score and micro F-score and what different functions and implications do they have?

The biggest difference between the two is the fact that the macro F1 score is capable of returning accurate measures of performance when classes are imbalanced, which is not really the case with the micro F1 score.

References

L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. 1984. *Classification and Regression Trees*. Taylor & Francis.

Leo Breiman. 2001. Random forests. *Machine Learning*, 45(1):5–32.

Yu-Jing Chang Chen-An Tsai. 2023. Efficient selection of gaussian kernel svm parameters for imbalanced data. *Genes*.

Corinna Cortes and Vladimir Vapnik. 2009. Support-vector networks. *Chem. Biol. Drug Des.*, 297:273–297, 01.

Chen B.-Bernstein M. S. Fast, E. 2016. Empath. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*.

Fix and Hodges. 1985. *Discriminatory analysis: nonparametric discrimination, consistency properties*, volume 1. USAF school of Aviation Medicine.

David D. Lewis. 1998. Naive (bayes) at forty: The independence assumption in information retrieval. In Claire Nédellec and Céline Rouveirol, editors, *Machine Learning: ECML-98*, pages 4–15, Berlin, Heidelberg. Springer Berlin Heidelberg.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

J. Ross Quinlan. 1986. Induction of decision trees. *Machine Learning*, 1:81–106.

J.R. Quinlan. 1987. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234.

J Ross Quinlan. 1993. Program for machine learning. *C4. 5*.

Irina Rish. 2001. An empirical study of the naïve bayes classifier. *IJCAI 2001 Work Empir Methods Artif Intell*, 3, 01.

Lior Rokach and Oded Maimon, 2005. *Decision Trees*, pages 165–192. Springer US, Boston, MA.

Vladimir Naumovich Vapnik. 1963. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780.