Assignment 3: Analysis and Parameter Search for Sentiment Classification with SVM

Vilém Zouhar
s5000076
vilem.zouhar@gmail.com

Edu Vallejo Arguinzoniz
s5016894
vallezoniz@gmail.com

Abstract

In this short report, we focus on using Support Vector Machines for binary sentiment classification of e-commerce reviews. We explore the parameter space via an exhaustive grid-search and comment on the implementation differences between the models `SVC` and `SVCLinear` provided by Scikit-Learn. We also focus on error analysis and the contribution of individual features in the vectorizer on the model decision. Lastly, we examine the "confidence" of the model and craft an adversarial example by adding noise to the review, demonstrating the fragility of the classifier.

1 Introduction

The goal of sentiment analysis is to assign sentiment and possibly its intensity to usually a paragraph of text. In the simplest form, it is a classification of texts into either *positive* or *negative* categories. Support Vector Machines classifiers (Cortes and Vapnik, 1995) have been widely used for this task (Ahmad et al., 2017; Ahmad et al., 2018). Both Pranckevičius and Marcinkevičius (2017) and Hasanli and Rustamov (2019) empirically demonstrate that the performance of SVMs is similar to those of naive Bayes or decision trees. Logistic regression, however, appears to perform slightly better (Poornima and Priya, 2020).

In this report, we perform grid-search to find the best model parameters and explore their effects on model performance. We follow up with a discussion on the implementation difference between the various parameters within SVM. Finally, we bring our attention to a detailed analysis of individual feature contribution to the classification, model confidence and crafting an adversarial example through noise injection. The data used in the experiments of this report are identical to those used in assignment 1 (Zouhar and Arguinzoniz, 2021) and we, therefore, skip its analysis and description. We nevertheless include examples in Section 4.

We introduce the evaluation methodology and models in Section 2. In Section 3 we discuss the performance and parameters of the best model. Error analysis together with measuring confidence and crafting an adversarial example is in Section 4. We conclude in Section 5.

Distribution of work. Vilém organized the document (abstract, introduction, summary, readme) and performed error analysis (features, confidence, adversarial example). Edu performed a large scale grid search, inspection of model differences and wrote the results section together with the analysis of parameters. Overall the work was distributed evenly and both author concur with this.

2 Methods

2.1 Evaluation

For parameter search, we use 10-fold cross-validation and for error analysis a single test set, consisting of 10% of the data (600 out of 6000 samples). For the implementation we make use of Scikit-Learn (Pedregosa et al., 2011), version 0.24.2.

2.2 Models

As mentioned previously this work focuses solely on Support Vector Machines. Despite this constraint, SVMs have many details to configure that can dramatically change the nature of the model. We thus focus on exploring SVMs under different configurations.

2.2.1 Linear vs non-linear

The first decision variable we considered is the type of SVM kernel. We are particularly interested if the kernel is (1) **linear** or (2) **non-linear**. We make this separation as a linear kernel implies that linear solvers can be used to optimize the SVM parameters, which not only should be faster but also more flexible as they allow for different loss and regularization functions. In fact, Scikit-Learn defines two completely different classes to represent linear (`LinearSVC`) and non-linear (`SVC`) SVM classiefiers with different hyperparameters to tune in each.

2.2.2 Kernels

The next factor we consider is the kernel function. This parameter only affects non-linear SVMs as the linear SVM has a fixed linear kernel function. There exists a rich variety of SVM kernels and Scikit-Learn even lets us define our own by passing a callback to the SVM estimator. Although an interesting topic, this work has limited scope and does not contemplate the in-depth mechanics of SVMs and as such, this is left out.

Instead, we make use of the kernels already provided by the library which also happen to be the most popular. There are four in total: **linear** (a special case of polynomial kernel), **polynomial**, **RBF** and **sigmoid**. For the polynomial kernel, we can also dictate the degree of the polynomial which will result in kernels that behave very differently (even if they are all polynomial), we try out polynomial kernels of degree 2, 3, 4 and 5.

2.2.3 Regularization

Finally, a common hyperparameter across all SVMs, no matter the kernel, is the regularization coefficient C. Regularization is a common technique in Machine Learning that consists in limiting the learning capacity of a model to avoid the model from fitting the train data too tightly and thus avoiding overfitting.

Regularization can be done in many different ways, in the case of SVMs, it is typically implemented as a penalty term in the optimization target objective function. This term is usually a metric of the parameters of the SVM that penalizes parameter "complexity". Norm functions (distance from the origin) are usually chosen as the penalty metric, some examples are the $\boldsymbol{L^1}$ **norm** (also known as **Manhattan Norm**) and the $\boldsymbol{L^2}$ **norm** (or **Euclidean Norm**).

The regularization coefficient C scales the strength of the regularization term. For testing purposes, we chose evenly spaced regularization terms in a logarithmic scale starting from 2^{-4} until the maximum of 2^4.

2.2.4 Classification vs Regression

SVMs can be used for both classification and regression, and although this task is proposed as a clear classification task we can reduce it to a regression task. To do this we assign numerical values to the labels, e.g. 1 for positive and 0 for negative. We then train the regressor SVM as if it were a regression task.

When it is time to test we can bucket the continuous values predicted by the SVM to the values that represent the labels with an arbitrary threshold. Because we chose the numerical values 0 and 1, we consider the threshold 0.5, i.e. *if* $x \geq 0.5$*:* POSITIVE *else* NEGATIVE. In case we chose the values as -1 and 1, then the threshold would be naturally 0. This way we can obtain discrete predictions and compute an accuracy score which can be compared to the classifying SVM.

As interesting as this approach appears, it unsurprisingly yielded marginally the same results as the classifying SVM counterpart, and we will therefore not report the results explicitly. As a curiosity, we observed that training regression SVMs was about 20% faster regularly but it had problems converging on occasions and a single fit could take up to 3 minutes in such cases (compared to 30 seconds on average for SVC).

3 Results

In the following subsections, we report the effect of changing various model parameters on the task accuracy, evaluated using 10-fold cross-validation. We consider maximum number of features, ngram range, filtering of stopwords, C, γ and the kernel.

3.1 Parameter C

To find out the effect that the hyperparameter C has with different kernels we plot the accuracy of the SVMs against C in Figure 1

In the figure we can observe two behaviours, SVMs with some kernels (RBF and polynomial) perform very poorly with low C but plateau even with very high C, while SVMs with other kernels (linear and sigmoid) perform respectably in a wide range of C values and do not plateau, meaning

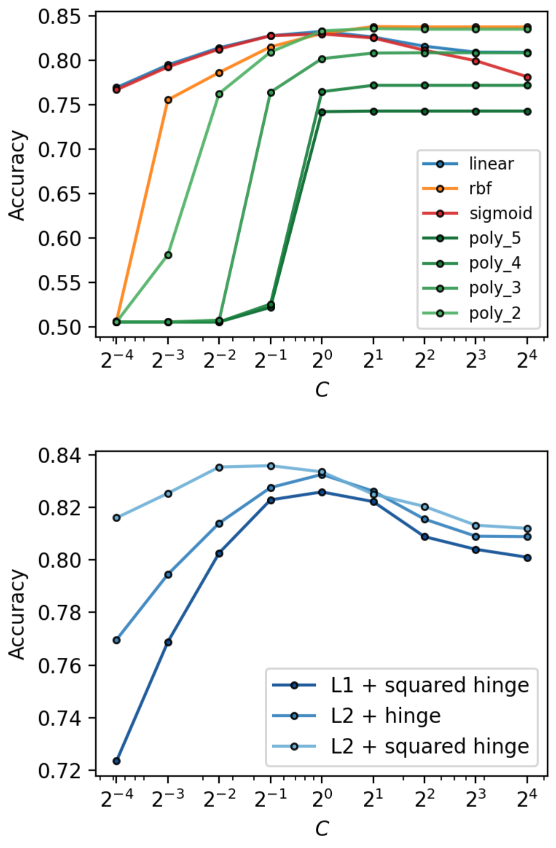

Figure 1: Value of regularizing coefficient C against the accuracy of an SVM with non-linear kernels (top) and linear kernel (bottom).

they have a clear optimal C value and their performance decreases past this optimum.

Although not quite optimal for all kernels, the value $C = 1$ appears to be a good choice (which is the default in Scikit-Learn). All other kernels except the polynomial kernels of degree three or higher show similar performance with the right choice of C.

3.2 Features Count

We show the relationship between the number of used features by the vectorizer (parameter `max_features`) and the performance. Again, across all parameters, filtering stopwords does not add to the performance. The performance also does not improve significantly over 5k features being used by the model.

3.3 Best model

The best accuracy was 0.86 and was obtained with the RBF kernel, $C = 2$, TF-IDF vectorizer with 100k max features, ngram range `(1,3)`,

Linear + default		True class	
		Positive	Negative
Pred.	Positive	244.0	47.6
	Negative	52.8	255.6

Best model		True class	
		Positive	Negative
Pred.	Positive	252.2	40.6
	Negative	44.6	262.6

Table 1: Confusion matrix of the linear model with default parameters and the best model using 10-fold cross validation

	Precision	Recall	F_1
Linear + default	0.84	0.82	0.83
Best model	0.86	0.85	0.86

Table 2: Precision, recall and F_1-score (for the positive class) of the default and best model using 10-fold cross validation

no stopwords filtering and $\gamma = 0.6$. The associated F_1 scores is 0.86 (both macro- and micro-averaged). A complete per-class breakdown is presented alongside the confusion matrix in 1 and 2. The best model is put in contrast with the linear model with default parameters (and TF-IDF vectorizer) which achieved 0.86 accuracy and 0.85 F_1-score.

The effect of changing γ was marginal, though systematic with a $\frown$ shape in relation to the performance, with peak at $\gamma = 0.6$. Similarly to assignment 2, neither stemming nor adding extra manual features such as review length, supported by findings in Section 4, did not improve the model.

3.4 Training time

We now look at the training time of the SVMs with different kernels. In Figure 2 we show boxplots that summarize the distribution of training times for each kernel.[1]

We measure the distribution of cv averaged training times produced by different values of C.

[1]The experiments were run on a server running the Linux kernel release `5.14.7`. The relevant hardware of this server consists of an Intel i9-9900KF processor running an all-core 5.0 GHz overclock, 64GB of DDR4 RAM at 3600Mhz (cl18) and a Samsung 970 EVO NVMe SSD (hosting an extra 64GB swapfile).

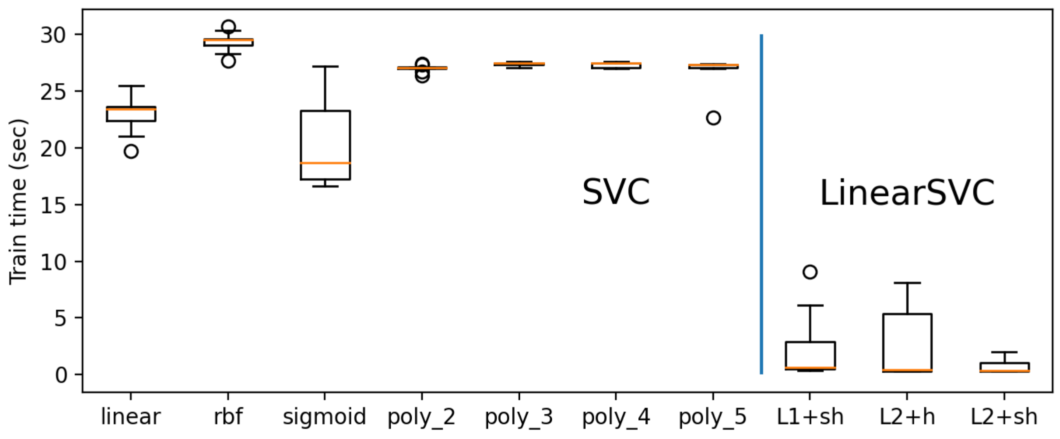

Figure 2: Distribution of training times for each kernel configuration.

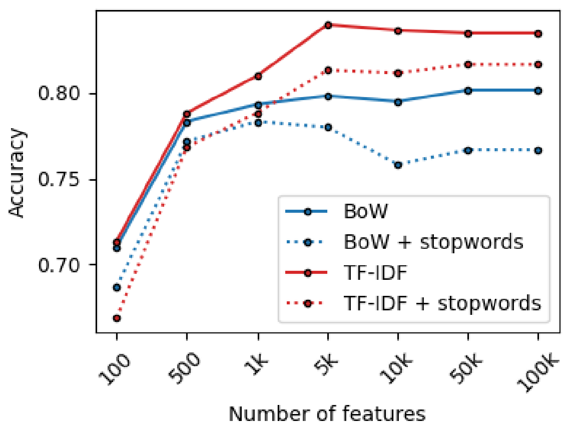

Figure 3: Feature count in vectorizer and model performance (SVM with linear kernel). Evaluated on 10% of the data.

This approach yields more interesting results as the standard deviation across folds was very low within the same configuration, and the C parameter can affect convergence with some kernels resulting in longer training times. The vectorizers were re-fit each fold adding a small overhead that does not correspond to the train time of the SVM.

We can immediately see the speed gap between SVMs with linear kernels implemented by LinearSVC and the SVMs with non-linear kernels implemented by SVC. Even when we compare the linear kernel that SVC implements the gap does not narrow. The reason for the gap stems from the fact that SVC is implemented for all sorts of kernels (it uses `libsvm`) and cannot make any assumptions about it, because of that it needs to use general parameter optimization algorithms that work with any kernel. On the contrary, LinearSVC only works with a linear kernel, and can thus use efficient linear solvers (from `liblinear`) to speed up the parameter search.

Because of the comparable performance of the linear to other kernels and the speedup in training, we advise to use this model. This may not be immediately transferrable to other tasks and data and as such, careful testing should be always done.

Besides this gap between the two implementations, the train times are quite packed within each implementation. Some kernels do however show more variance than others in train time, we can attribute this to the number of optimization steps that are not consistent across runs and thus lengthen some training times.

3.5 Training Data Size

In Figure 4 we show the relation between available training data and performance together with different vectorizers and stopword filtering. Based on the graph, we can conclude that providing more training data would still lead to considerable improvements in performance. We can also see that TF-IDF is by far the best vectorizer and that filtering stopwords do not add to the performance for SVM. This is in contrast with the previous models, especially Naive Bayes.

4 Error Analysis

Similarly to the previous report, we examine whether the length of the article is a good predictor for whether it will be misclassified. In contrast to the results for Naive Bayes and Logistic Regression, this does not appear to be the case with SVM, as is shown in Figure 5. In fact, no systematic rela-

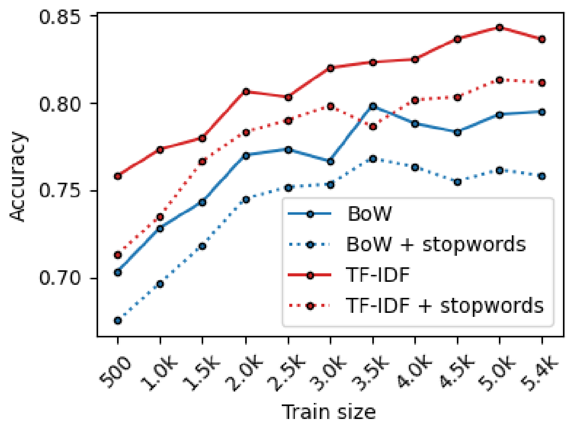

Figure 4: Training data size and model performance (SVM with linear kernel). Evaluated on 10% of the data.

tionship appears to be present in terms of misclassification. This is further supported by the fact that the average review length for correct and incorrect classification is 156 and 161. Results for the other vectorizer, Bag of Words, are similar though with higher variance (not shown). The lengths for correct and incorrect classification when using this vectorizer are 156 and 160, respectively. This suggests that the vectorizer has very little impact on the distribution.

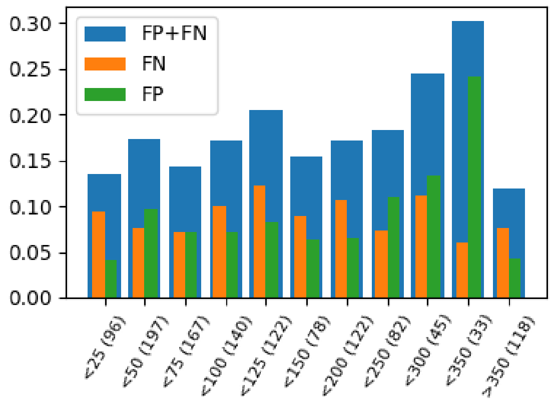

Figure 5: Dependency of review length (in tokens) on misclassification rate (provided by default SVM with linear kernel with TF-IDF with 10k features). Numbers in brackets show bucket size.

4.1 Feature Contribution

The trained SVM parameters can be examined with respect to the coefficients corresponding to individual positions in the input vector - tokens. The higher the coefficient, the more it will move the computation above the hyperplane (classify as positive) and the lower the coefficient, the more it will move under the hyperplane (classify as negative). In Figure 6 we select the top 8 most positive, top 8 most negative and 2 neutral tokens based on these coefficients. We also distinguish between using individual tokens or n-grams as the input. Based on these coefficients, we get a reasonable split between positive and negative tokens and n-grams. Surprisingly this also includes phrases that are not negative in themselves, such as *none of*, *does not* or *is not*. This is caused possibly by the higher use of these phrases for negative reviews.

The Pearson's correlation coefficients between the coefficients of models trained either with Bag of Words or TF-IDF is 0.79. This correlation decreases with a higher number of features. For 100 features, it is 0.93 but for 100k features, it drops to 0.72, presumably due to more noise, especially on low-frequency terms.

The coefficients for Bag of Words are much lower than for TF-IDF (average of absolute value is 0.12 and 0.30 respectively). This is caused by the vectorized inputs being of different magnitudes. While TF-IDF vectorizer produces normalized vectors to L^2-norm 1.0, the vectors from Bag of Words are naturally much higher, on average 20.4 L^2-norm.

Examples. Finally, we list several correctly and incorrectly classified examples with each word color being either positive ($c \geq +0.5$), negative ($c \leq -0.5$) or neutral. The examples are capitalized and detokenized to increase readability. The examples are outputs of the default SVM classifier model with a linear kernel with Bag of Words vectorizer.

„Many albums, including this one , have the problem that all the songs pretty much sound the same; but this also has the troublesome problem of music that is too easy on the ears and that never gets very exciting - except for the first two tracks. The use of many poetry elements in their lyrics ca n't save the blandness of the music. "What about everything" and "Life less ordinary" are good songs. The rest pretty much sound the same and will put you to sleep"
→ **Negative (-1.00)**

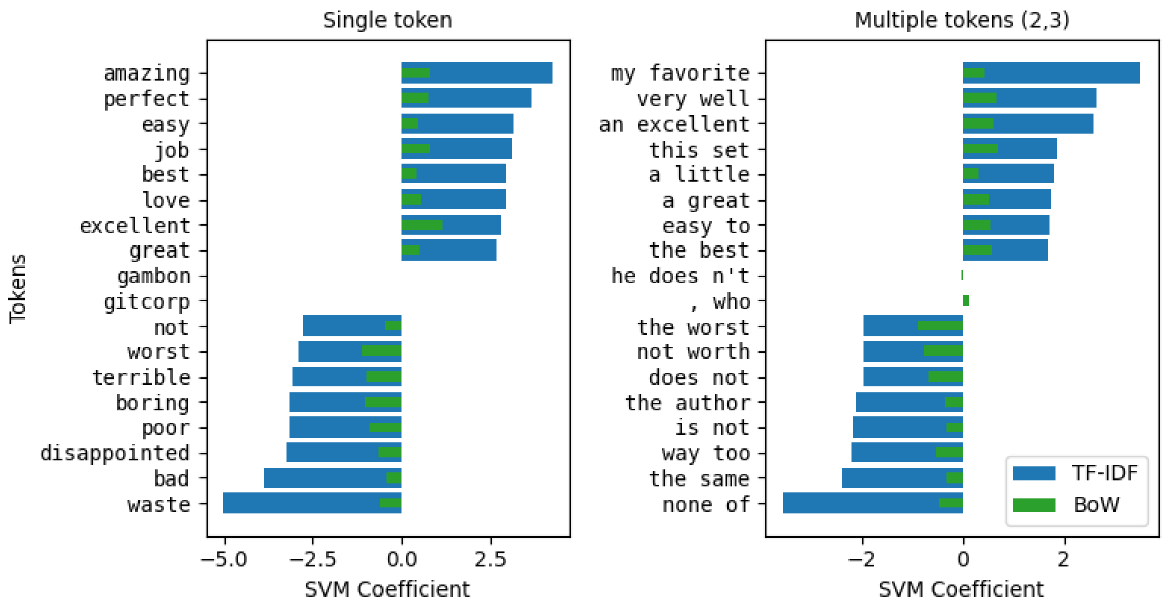

Figure 6: Comparison of tokens (left) and n-grams ($n \in \{2, 3\}$, right) with positive, neutral and negative effect on sentiment. Coefficients are provided by a SVM classifier with linear kernel and otherwise default parameters and either Bag of Words or TF-IDF vectorizers, both restricted to 10k features.

„*This camera works well , except that the shutter speed is a bit slow. The image quality is decent. The use of aa rechargeable batteries is also convenient. The camera is pretty sturdy. I've dropped it a few times and it still works fine*"
→ **Positive (+2.58)**

Both examples were correctly predicted despite the quite high degree of noise and number of conflicting words.

4.2 Confidence

Compared to simpler models, like Naive Bayes or Logistic Regression, the SVM classifier does not have a way to compute confidence, a number in $[0, 1]$. It can, however, be estimated by the distance from the decision boundary (hyperplane). The reason this can't be used straightforwardly is that it's possibly unbounded in the interval $[0, \infty)$. The numbers in the positive example show this quantity (negative for negative examples). Through this report, we refer to this quantity as confidence, though formally it does not fulfil the standard properties. The scikit implementation of SVM classifiers supports[2] outputting probabilities though this increases training times and, more importantly, creates additional steps in the computation using Platt scaling (Platt, 1999), which makes the interpretation more complex.

The average confidence (absolute values) does not differ much between test and training examples but depends on whether BoW or TF-IDF is used. Section 4.2 depicts this phenomenon. The increased number of features used in the vectorizer has a slightly negative impact on the confidence (not shown).

	BoW	TF-IDF
Train	2.21	1.03
Test	2.25	0.92

Table 3: Average confidence during inference on train and test with both vectorizers (10k features).

The average confidence across the confusion matrix is shown in Section 4.2. A good train of models with both vectorizers is that they have higher confidence in correct results. This is pronounced more for the model with TF-IDF. There seem to be few systematic differences between the individual classes (FP-FN and TP-TN).

4.3 Adversial examples

Since the scoring in the SVM classifier is additive with respect to the tokens, it is fairly easy to man-

[2]scikit-learn.org/stable/modules/svm.html#scores-probabilities

BoW		True class	
		Positive	Negative
Pred.	Positive	2.12	1.19
	Negative	1.41	2.34

TF-IDF		True class	
		Positive	Negative
Pred.	Positive	1.04	0.37
	Negative	0.38	1.08

Table 4: Average confidence in confusion matrix on all data (train+test) with both vectorizers (10k features).

ually craft an adversarial example by adding otherwise benign tokens. This is made permissible by the fact that the majority of tokens have non-zero coefficients even though semantically they are neutral. For Bag of Words, this issue is more problematic because one extra occurrence will become an unscaled addition of the coefficient which moves the computation closer to the decision boundary.

We list the coefficients for tokens we consider to be otherwise benign and may be less noticeable (small and composed of non-alphanumeric characters) in Section 4.3. Interesting is the distinction in polarity between `(`,`)` (left and right bracket) for which we are unable to provide any explanation.

Symb.	Coef.	Symb.	Coef.
`...`	-0.13	`..`	-0.02
`.`	+0.01	`:`	-0.07
`,`	-0.03	`` ` ``	-0.01
`/`	+0.02	`-`	-0.03
`"`	-0.06	`%`	+0.12
`!`	-0.01	`#`	-0.24
`&`	+0.05	`+`	+0.01
`;`	+0.01	`--`	+0.01
`?`	-0.06	`=`	+0.06
`(`	-0.10	`@`	-0.09
`)`	+0.17	`~`	-0.03
`'`	+0.01	`*`	-0.07
`$$$`	-0.22	`$`	+0.00

Table 5: Coefficients of small (≤ 3) tokens composed of non-alphanumberic characters based on linear SVM with BoW vectorizer

When considering the second example of the previous section, adding the token # (hashtag) 11 times changed the correct and confident prediction of the model from positive to negative. The coefficient of this symbol was -0.24. Misclassification occurs at the 11th addition, because 11 is the smallest nataural k such that $2.58 + k \cdot (-0.24) < 0$.

> *„This camera works well, except that the shutter speed is a bit slow. The image quality is decent. The use of aa rechargeable batteries is also convenient. The camera is pretty sturdy. I've dropped it a few times and it still works fine # # # # # # # # # # #"*
> $\rightarrow$ **Negative (-0.04)**

For TF-IDF, especially when the term frequency is log-scaled, this attack becomes less and less effective and the previous example required more added noise to become misclassified. This sort of attack is very common and explored for simple models such as Logistic Regression, Naive Bayes and SVM, especially in the context of spam detection (Jorgensen et al., 2008). It can even be crafted automatically with projected gradient descent (Wang et al., 2021). A viable defence for SVM would be to prune coefficients of low magnitude. This regularization would also be a way to prevent overfitting.

An attack against this defence would be to use modifiers, which usually have high coefficients, in combination with a negative modifier, such as *„It is super super super great at being bad."* To combat this, more complex measures would need to be applied.

5 Summary

We examined the parameter space of SVM as provided by Scikit-Learn and found that the best parameters are close to the ones provided by default and that fine-tuning them provided only a small improvements. The best model ($C = 2$, RBF kernel, TF-IDF vectorizer, 100k features with ngram range `(1,3)`, $\gamma = 0.6$) was cross-evaluated to 0.86 accuracy.

We also concluded that the model could benefit from access to more data and that, interestingly, there is little relation between review length and prediction accuracy (as is the case with naive Bayes and logistic regression).

The analysis of feature coefficients of single tokens and n-grams yielded results that were in accord with our intuition (i.e. classification of positive and negative phrases). We also demonstrated the ease of an adversarial attack by adding noise in the form of 11 single-character tokens on SVM with Bag of Words vectorizer.

Future work For more detailed error analysis, one could train another model that predicts whether the SVM will make a correct classification. Such a model, if useful, would actually be useful in determining e.g. whether a human needs to be put in the loop for a specific case. A comparison could be made across various model families (e.g. SVM, Logistic Regression or Naive Bayes) together with the baseline self-reported model confidence.

References

[Ahmad et al.2017] Munir Ahmad, Shabib Aftab, and Iftikhar Ali. 2017. Sentiment analysis of tweets using svm. *Int. J. Comput. Appl*, 177(5):25–29.

[Ahmad et al.2018] Munir Ahmad, Shabib Aftab, Muhammad Salman Bashir, Noureen Hameed, Iftikhar Ali, and Zahid Nawaz. 2018. Svm optimization for sentiment analysis. *Int. J. Adv. Comput. Sci. Appl*, 9(4):393–398.

[Cortes and Vapnik1995] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning*, 20(3):273–297.

[Hasanli and Rustamov2019] Huseyn Hasanli and Samir Rustamov. 2019. Sentiment analysis of azerbaijani twits using logistic regression, naive bayes and svm. In *2019 IEEE 13th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–7. IEEE.

[Jorgensen et al.2008] Zach Jorgensen, Yan Zhou, and Meador Inge. 2008. A multiple instance learning strategy for combating good word attacks on spam filters. *Journal of Machine Learning Research*, 9(6).

[Pedregosa et al.2011] 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.

[Platt1999] John C. Platt. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods.

[Poornima and Priya2020] A Poornima and K Sathiya Priya. 2020. A comparative sentiment analysis of sentence embedding using machine learning techniques. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 493–496. IEEE.

[Pranckevičius and Marcinkevičius2017] Tomas Pranckevičius and Virginijus Marcinkevičius. 2017. Comparison of naive bayes, random forest, decision tree, support vector machines, and logistic regression classifiers for text reviews classification. *Baltic Journal of Modern Computing*, 5(2):221.

[Wang et al.2021] Chenran Wang, Danyi Zhang, Suye Huang, Xiangyang Li, and Leah Ding. 2021. Crafting adversarial email content against machine learning based spam email detection. In *Proceedings of the 2021 International Symposium on Advanced Security on Software and Systems*, pages 23–28.

[Zouhar and Arguinzoniz2021] Vilém Zouhar and Edu Vallejo Arguinzoniz. 2021. Assignment 1: Comparison of baseline models for sentiment and topic classification. Not published. Learning from Data assignment.