Abstract

In this paper I present two classifiers, an SVM/SVC and Decision tree classifiers. These classifiers are deployed to solve a text classification problem. The problem involves using user reviews and ratings to classify the severity of the side effects of a drug, on a five-step scale. The model is trained by using a dataset sourced from druglib.com. I present three sets of results, which show the differences between using different methods to optimize the results. The results in this paper indicate that although the Decision tree performed better for the last set of results. It performed worse than the SVM on the other sets of results. The Decision tree by its nature Is unreliable. While the SVM is much more reliable. This is proven by the results and previous research done on SVM and Decision tree classifiers. For this reason, I conclude that the SVM is a better classifier for this problem. I also discuss possible further areas of research including using a Logistic Regression classier, different POS tagging and Lemmatizers, etc., which could possibly improve the results.

Introduction

Drugs are often used to treat various conditions including diseases and mental illness. However, the usage of drugs can produce negative side effects. The side effects may have different levels of severity. The ability to predict the severity of side effects could be extremely important. It could be useful to various individuals and groups, such as drug researchers, pharmaceutical companies, and public health institutions. In this paper I propose two text classification models (as well variations of the models). One using a Support Vector Machine (SVM) and the other using a Decision Tree. The dataset is sourced from druglib.com and contains 8 attributes.

There has been a significant amount of research done in the text classification involving user generated reviews. There also has been research done in classifying the severity of side effects. Proving that there is a strong interest in this topic. Text classification is a difficult problem to solve. This is because computers find understanding text extremely difficult. Therefore, Nature Language Processing (NLP) is extremely important. NLP can help enable computers to understand contextual language and the semantic meanings of words within text.

I collect three sets of results for both classifiers. Each set represents a variation of the model, with various text processing, parameters, and other methods applied to the data and model. These methods are applied to optimize the results. In this paper, I discuss these results and compare the two classifiers. I also discussed the impact of the different methods used to improve the results, such as hyperparameter optimization, feature engineering and different text processing methods. I also outline more possible areas of further research.

Data Description

The data used in this paper is sourced from druglib.com. Which was compiled and made freely available online for research purposes only, on the UCI Machine Learning Repository (Kallumadi & Gräßer, 2018). The dataset is provided in the tab separated columns format (.test). The file read in using the python library pandas.

The dataset contains 4143 instances. 3107 (75%) records of the dataset are used for the training set. And the remaining 1036 (25%) records are used for the testing dataset.

There are 8 attributes in the dataset. (Shown in the table below)

Attribute	Description	Type
urlDrugName	The name of the drug	Categorical
condition	The name of condition	Categorical
benefitsReview	Patient's review of the benefits of the drug	Text
sideEffectsReview	Patient's review of the side effects of the drug	Text
commentsReview	Patient's overall comments of the drug	Text
rating	Patient's rating (1-10)	Numerical
sideEffects	Patient's side effects rating (No Side Effects, Mild Side Effects, Moderate Side Effects, Severe Side Effects, Extremely Severe Side Effects)	Categorical
effectiveness	Patient's effectiveness rating (Ineffective, Marginally Effective, Moderately Effective, Considerably Effective, Highly Effective)	Categorical

This paper will mainly focus on the sideEffects and the three review columns (benefitsReview, sideEffectsReview and commentsReview). However, the effectiveness and rating columns will also be used in training the model.

Approach

In this paper I will present three sets of results, which have been modified to get better results. Each set of results have three trials per classifier. This means that there is total of 6 outputs per sets of results. The first results contain the results of the model using only the three review columns, benefitsReview, sideEffectReview and commentsReview to perform sentiment analysis, with stemming, lemmatization, vectorization done to the text data. The second results are the results with rating and effectiveness added as features to the model. There is also grid search (a type of hyperparameter optimization) done to improve the results of the classifiers. The third set of results has POS tagging implemented, it also uses TFID vectorizer instead of Count Vectorizer, and the stemming process was removed. The classifiers' goal is to determine the severity of the side effects. As in the classifier gives a classification of either No Side Effects, Mild Side Effects, Severe Side Effects and Extremely Severe Side Effects. Two classifiers were used, Support Vector Classification (SVC) classifier and Decision Tree classifier (DT). The weighted average for the precision scores, recall score, f values and accuracy scores are collected. The precision score is the ratio of correctly classified positive results to the total positive classified results. Recall score is the ratio correctly classified positive results to all results. F-value (f1 value) is the weighted average of recall and precision score. Accuracy Score is the ratio of correct results. Since the data is unbalanced f value would be the most important metric for determining model's overall performance. The basic methodology for the text mining and the text mining process used in this paper is as follows.

- Text Preprocessing

- Vectorization
- Classification

Text Preprocessing

The three review columns benefitsReview, sideEffectReview and commentsReview are combined into one review column. This allows for easier application of text preprocessing methods.

Tokenization is performed on the review column; this involves converting the text to a list of tokens. Tokens are sequences of characters. In the English language they are usually separated by spaces, hyphens, etc. All tokens are case folded by converting them to lowercase. After the column is tokenized, case-folded and punctuation removed. The stopwords will be removed from the tokens. Stopwords are words that have low sematic information, these include words such as “is”, “like”, “could”, etc. After the stopwords are removed the remaining tokens are known as terms. The terms are then POS (part of speech) tagged. POS tagging provides context for words, by providing an identification such as a noun, adverb, adjectives, etc. This can improve the results when lemmatizing the words.

Lemmatization is then performed on the lists of terms. Lemmatizing convert the terms into lemmas, i.e. the dictionary form base word. It usually does this by removing the inflections at the end of the words. This means that they can be processed as single term. For example, the lemma for “talk”, “talks”, “talking”, “talked” would be talk. For the first two sets of results Stemming was used. Stemming is similar to lemmatization. However, its main difference is that it converts the terms into a word stem rather than the dictionary form of the word. This means that the word may not be real words. For example, the word stem for “talk”, “talks”, “talking” and “talked” will be “tal”. The algorithm used for stemming is the Porter’s Algorithm. Which is an aggressive algorithm and is the highest performing stemming algorithm. However, it slows down the text processing process. (Porter, 1980)

Vectorization

In-order for the text to processed by a classifier, the text needs to be vectorized. The first two set of results use a Count Vectorizer while the last set of results uses a TFIDF vectorizer. Vectorization in NLP is the process of converting pieces of text into an array of numbers. In this paper I have limited the amount of words from the corpus to 150. For the Count Vectorizer each row has an integer value for all 150 of the features. This means that the training set’s matrix size is 3107x150 and testing set is 1036x150. Each value represents how many times that word appears in the text, this value is called term frequency. The TFIDF vectorizer does not just keep track of the term frequency. A TFIDF (Term frequency inverse document frequency) value, rather than an integer number is a floating-point number, which combines the term frequency with inverse document frequency (IDF). The IDF represents how common or rare the term is in relation to the rest of the document. Both values are combined, the TFID represent more information than just term frequency. The two categorical attributes that I am using, effectiveness and sideEffects are also converted to numbers. Since they are not a text attribute, we can just convert the corresponding category to a number ranging from 0-4. With addition of the effectiveness and rating, there is a total of 152 features.

Classification

Support Vector Classification

A Support Vector Classification (SVC), not to be confused with Support Vector Clustering, is a SVM classifier that used in Machine learning which can be used for regression as well as classification. It

involves a creation of creating a hyperspace or multiple hyperspaces, which are used for the classification. It works by mapping the vectors of the features to high dimension feature space. (Cortes & Vapnik, 1995) This SVC is based on libsvm an open library of Support Vector Machines (SVM) according to the sklearn documentation (sklearn.svm.SVC, n.d.). SVM is also one of the most popular classifiers for text classification. SVM are usually have a linear kernel. However, using the kernel trick

Decision Tree

A Decision Tree classifier uses a classification tree in-order to classify data points. A Decision Tree takes in a vector of features. Each node except for the leaves in the tree represents a single feature. The leaves represent the classes. So, in this case they will represent the five side effects categories. The concept behind a decision tree is it learns to the predict the target variables by learning rules based on the features (tree, 2020).

Hyperparameter Optimization

To improve the results for the second set of results, a grid search was used in-order to find more optimal parameters. A grid search is an exhaustive search that will test all possible configurations and combinations of all the parameters provided. Regarding the SVC the parameters that I attempted to optimize were C, gamma coefficient, kernel, and class weights. C is a regularization parameter, the values that I inputted into the grid search were 1, 10, 100, 1000. The Gamma coefficients I used were 1, 0.1, 0.01, 0.001, 0.0001 and auto. For the kernel I inputted linear and Radial basis function (RBF). For class weights, balanced and none. The optimal parameters I found were C being 1, gamma being auto, RBF kernel and class weights are balanced. For the decision tree I used grid search on two parameters, max depth, and random state. For max depth I tried all depths ranging from 3-10. For random state I used, 0, 1 and none. The optimal parameters I found were, a max depth of 6 and a random state of 1.

Results

Table 1: Results 1

Classifier	Trial Number	Weighted Average Precision Score	Weighted Average Recall Score	Weighted Average F-Values	Accuracy Score
SVM	1	0.28	0.33	0.27	0.33
SVM	2	0.28	0.33	0.27	0.33
SVM	3	0.28	0.33	0.27	0.33
SVM Average		0.28	0.33	0.27	0.33
Decision Tree	1	0.20	0.33	0.22	0.33
Decision Tree	2	0.20	0.33	0.22	0.33
Decision Tree	3	0.20	0.33	0.22	0.33
DT Average		0.20	0.33	0.22	0.33

Table 2: Results 2

Classifier	Trial Number	Weighted Average Precision Score	Weighted Average Recall Score	Weighted Average F-Values	Accuracy Score
SVM	1	0.49	0.49	0.48	0.49
SVM	2	0.49	0.49	0.48	0.49
SVM	3	0.49	0.49	0.49	0.49
SVM Average		0.49	0.49	0.49	0.49
Decision Tree	1	0.44	0.45	0.44	0.45
Decision Tree	2	0.44	0.45	0.44	0.45
Decision Tree	3	0.44	0.45	0.44	0.45
DT Average		0.44	0.45	0.44	0.45

Table 3: Results 3

Classifier	Trial Number	Weighted Average Precision Score	Weighted Average Recall Score	Weighted Average F-Values	Accuracy Score
SVM	1	0.50	0.50	0.50	0.50
SVM	2	0.50	0.50	0.50	0.50
SVM	3	0.50	0.50	0.50	0.50
SVM Average		0.50	0.50	0.50	0.50
Decision Tree	1	0.52	0.52	0.51	0.52
Decision Tree	2	0.52	0.52	0.51	0.52
Decision Tree	3	0.52	0.52	0.51	0.52
DT Average		0.52	0.52	0.51	0.52

Discussion

Since our data is imbalanced, the f-value would be a good metric over the other metrics. It takes in account, both the precision and recall so it is a better metric than accuracy for showing overall performance of the models.

As we see in table 1 the SVM performs much better than the Decision Tree classifier. We see that the average precision score for the SVM is 0.28 compare the Decision Tree which has an average precision score of 0.20. They both have an average recall score of 0.33. SVM has higher F-value of 0.27 compared to 0.22. They both have an accuracy score of 0.33. The f value is higher for the SVM, this indicates that the SVM model performed better than the Decision tree model. These results show the impact of various techniques on the results. For the first results that SVM performed better than the Decision Tree classifier. I assume that the SVM worked better in the first set of results due to previous data on the performance of SVM and Decision Tree done by other researchers. For example, in a paper that

compared the differences in performance between various classifiers (including SVM and Decision tree) in classifier Amazon product review data. The researchers found that the Decision Tree performed the worst out all the classifiers being research. The researchers also found that a Logistic Regression model was the best model (Pranckevicius & Marcinkevičius, 2017). This would explain the reason for the Decision Tree performing worse than the SVM.

As we can see in table 2. The addition of effectiveness and rating as features along with the hyperparameter optimization significantly improved the results for both classifiers. SVM has a higher precision score of 0.49 compared 0.44 for the DT. SVM also has 0.49 for the recall score compared to 0.45. The f-value is 0.49 for SVM and 0.44 for the DT. SVM is also more accurate with an accuracy score of 0.49 compared to 0.45 for the DT classifier. The scores for the SVM are 0.4-0.5 higher than the DT's scores. This shows that the SVM model is still significantly better than the DT model. The second set of results shows the positive impact of feature engineering and hyperparameter optimization (through grid search) on the results. Through my experimentation and applying knowledge of the dataset and topic dataset I managed to find the best features for text classification. The text from the three review columns is important, as the patient may describe their side effects, this would be important for determining the severity of their side effects. I also added in the rating. The rating from experimentation has showed an increase in an accuracy score by 5-10% points. I assume that this because patient's rating is dependent on the severity of the side effects. So, if the side effects are more severe than the patient might be more inclined to give a lower rating and vice versa. Effectiveness also had a slight impact on the accuracy score with around an increase of 1-2% points. I assume this is because a patient may feel a drug is less effective if it gives more severe side effects. However, severity of symptoms is likely not a big factor in the decision of patient to rate the effectiveness of a drug, hence the low impact of effectiveness on determining side effects severity. There were two attributes that I did not include in my model, urlDrugName and condition. I did not pick urlDrugName as there is 502 unique drug names in the training set. For a dataset with only 3107 rows, this means that there is not much data per drug. This means that it would not be useful for classifying the severity of side effects. I also did not include condition. Although it is technically categorical data, it is manually entered by a patient. This means that there might be spelling errors, different grammar, spaces, etc. Which means that it would have to processed in-order to be useful. I also performed hyperparameter optimization. I used a grid search as describe earlier. However, since of hardware and time constraints the search was very limited. For the SVM only four different parameters were used in the search and 2-4 parameters were used (See Bottom of s A). However, the Decision tree does not have many parameters, so I am satisfied that I found the most optimal parameters for the Decision tree. If there was more time, then grid searching more parameters may lead to even more improvements.

Shown in table 3 is the third set of results. As we can see the addition of POS tagging and removal of stemming, only improved the results of SVM model by a slight margin. However, we can see that the decision tree improved by a relatively significant margin, the averages for precision score, recall score, f-value and accuracy score are all 0.50 for the SVM. For the DT, the averages for the precision, recall and accuracy score are 0.52 and the f-value is 0.51. These results indicate that the DT performed slightly better than the SVM. The removal of the stemming process and adding POS improved the results seems to show a significant change in the results. Interestedly, unlike the previous results the Decision tree, performed better than the SVM. A reason that possibly explain the better performance for both classifiers is that stemming algorithm used, Porter's Algorithm used might be too aggressive. This means that sometimes the Porter stemmer may process words into the same word stem. However, the words may have different semantic meanings. This may lead to poorer results. Using a less aggressive stemmer such as Snowball stemmer may help (Jivani, 2011). However, this will require

more research and experimentation in-order to determine if other stemmers are effective. The POS tagging also helped slightly, for the Decision Tree Classifier. However, I did not test other POS tagging methods. I only used the nltk POS tagger. There were some protentional issues with using this POS tagger. For example I had to convert the tags to wordnet tags in-order for it to work with the WordNet lemmatizer, this may of lead to some of the context and semantic meaning that the tags provided, to be lost. So further research needs to be done to investigate the impact of different lemmatizers and POS taggers.

A significant issue with the data is that it is imbalanced. The number of data categorized as No Side Effects is 930, Mild Side Effects is 1019, Moderate Side Effects is 614, Severe Side Effects is 369 and Extremely Severe Side Effects is 175. This data imbalance would be affecting our results. For the SVM I used balanced weights parameter, according to the sklearn documentation, this means that the weights are automatically adjusted inversely proportionately to the class frequency (sklearn.svm.SVC, n.d.). However, with the Decision tree the data needs to be balanced before being used, in the model. I did not balance the data. This mean that the results for the decision tree could be slightly improved if the data is balanced. The decision tree is also unstable as small variations in the data may cause an entirely different tree to be created, and therefore different results (tree, 2020). This could pose a problem with using the Decision Tree in further NLP models.

A look at the previous studies and research done on this specific dataset(druglib.com), also shows us some insights on possible ways to improve the results. In the original paper that the dataset was collected for (Gräßer, Kallumadi, Malberg, & Zaunseder, 2018). The researchers used two datasets one from druglib.com and the other dataset from drug.com. They did test their model in-domain (i.e. training on only the druglib.com dataset) they achieved an accuracy score 76.93% compared to my 50%(SVM) and 52%(DT). However, it is unfair to compare these results, as the researchers combined the side effects together, creating only three possible classes, No Side Effects, Mild/Moderate Side Effects and Severe/Extremely Severe side effects. This could be a useful technique to achieve better results and would increase my metrics for the classifiers. However. having only three classes as output, seems simplistic to me. In my opinion some of the nuance that the side effects levels originally had are lost. Also, if it is necessary to determine the differences between Mild/Moderate and Severe/Extremely Severe side effects, for whatever reason, then this method would not achieve desirable results. Another interesting factor of their research was that they used a Logistic Regression classifier, which was also used for the Amazon product review paper. This may indicate that the Logistic Regression classifier might be better for text classification involving reviews inputted by users. The paper also used a dataset to train the model, sourced from drug.com which had much more data, this led to better results when using that model. Another paper also used the druglib.com dataset along with the drug.com dataset, and they managed to get a misclassification rate of 27.1% (Dinh, Chakraborty, & McGaugh, 2020). However, they did use the three side effects categories as the other paper did. However, this still indicates that adding more data may improve my results.

I think judging by my results and the academic literature I think that the SVM classifier would be better to use than the Decision Tree classifier. Although my last set of results shows that the Decision tree performed slightly better on all the metrics I measured for. This is due to that fact that previous academic literature has stated that SVM performs better than Decision Tree. The sklearn documentation on the Decision tree states that the Decision is unstable and that changes in the variable can lead to completely different results. I think that the DT lack of stability is one reason for higher results for the last set compared to the relatively poor results of the other sets of results, in which SVM performed better than DT. SVM also scales better than DT. If I had to use one of these classifiers going forward, it would be SVM, for the reasons given prior.

Conclusion

In this paper I have proposed to two classification models. One using a SVM classifier and the other a Decision Tree classifier. Through experimentation I have found that through various techniques and processes that the Decision Tree performed slightly better, with it having a higher precision, recall, f value, and accuracy score. However, the results for the Decision Tree model, were usually lower until the last set of results. For this reason and the academic literature on the comparison of different classifiers, I have deemed these specific results as outliers, and therefore I would state that the SVM is better for determining the severity of the side effects of a drug. I have also outlined areas of possible further research. Such as using a Logistic Regression classifier, processing the condition column, and adding it as a feature, a more exhaustive grid search. Using different stemmers, using different lemmatization methods and POS taggers, balancing the dataset for the decision tree, lowering the number of outputs from 5 to 3 and lastly collecting more data from different sources such as drug.com and using that to train the model. All these areas of research could possibly lead to better results. Overall, my results show that the SVM model would perform more reliably than the Decision Tree classifier.

Reflection

I learned a lot about text classification and text mining. I found learning the differences between different methods, classifiers, algorithm incredibly useful. I found that research for this this assignment made me learn a lot more about the academic research done in text classification. There are several things that I would do differently if I were to do the assignment again. For example, I would look more into the areas of research, which I have outlined in the assignment. I would have picked different classifiers. I would of probably use the Logistic Regression classifier, seeing that recent research is using it more and more. I also felt that I spent too much time on coding the models. I felt that I tried to perfect the classifiers too much, and often tried things that lead to dead ends. For example, I initially tried using label encoding, which lead to poor results. I would also look it into using open source libraries for NLP such as spacy and Genism.

Bibliography

Ben-Hur, A., Horn, D., Siegelmann, H. T., & Vapnik, V. (2001). Support Vector Clustering. *Journal of Machine Learning Research*, 125-137.

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Journal of Machine Learning*, 273-297.

Jivani, A. G. (2011). A Comparative Study of Stemming Algorithms. *International Journal of Computer Technology and Applications*, 1930-1938.

Kallumadi, S., & Gräßer, F. (2018, October 2). Drug Review Dataset (Druglib.com). Retrieved August 19, 2020, from https://archive.ics.uci.edu/ml/datasets/Drug+Review+Dataset+%28Druglib.com%29

Porter, M. F. (1980, July). An algorithm for suffix stripping.

Pranckevicius, T., & Marcinkevičius, V. (2017). Comparison of Naïve Bayes, Random Forest, Decision Tree, Support Vector Machines, and Logistic Regression Classifiers for Text Reviews Classification. *Baltic Journal of Modern Computing, 5*, 221-232. doi:10.22364/bjmc.2017.5.2.05

sklearn.svm.SVC. (n.d.). Retrieved September 15, 2020, from scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Appendix A – Source Code for classification

Notes on Appendix A: The Stemming and hyperparameter optimization code are remained intact but have been commented out. Uncomment them to include them

```
import pandas as pd
import numpy as np
from nltk.tokenize import RegexpTokenizer
import nltk
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
stopwords = stopwords.words('english')
from nltk.corpus import wordnet
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()
#Remove the stopwords that have low semantic value
def remove_stop_words(text):
    words = [w for w in text if w not in stopwords]
    return words

#Turns all the tokens into lemmas
#Returns an array of text
def word_lemmatizer(text):
    lem_text = []
    for i in text:
        pos = i[1]
        #Convert the tag so wordnet lemmatizer can use it
        pos = convert_tag(pos)
        x = None
        #If no tag is specified just lemmatize without a tag
        if pos == '':
            x = lemmatizer.lemmatize(i[0])
        else:
```

```
            x = lemmatizer.lemmatize(i[0], pos=pos)
        lem_text.append(x)
    return lem_text

#Converts the nltk POS tags into POS tags that the wordnet lemmatizer can use
def convert_tag(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return ''

# Stem the text using the porter stemmer
def word_stemmer(text):
    stem_text = []
    for i in text:
        x = stemmer.stem(i)
        stem_text.append(x)
    return stem_text

#Convert the sides from categorical text to an int
#The models will out a number ranging from 0-4 corresponding to the respectie labels
def convert_side_effect_index(text):
    if text == "No Side Effects":
        return 0
    elif text == "Mild Side Effects":
        return 1
```

```
    elif text == "Moderate Side Effects":
        return 2
    elif text == "Severe Side Effects":
        return 3
    else:
        return 4

#Converts the effectiveness from categorical text to an int
def convert_effectiveness_index(text):
    if text == "Ineffective":
        return 0
    elif text == "Marginally Effective":
        return 1
    elif text == "Moderately Effective":
        return 2
    elif text == "Considerably Effective":
        return 3
    else:
        return 4

from nltk.tag import pos_tag
#Use the nltk POS tagger and gives each text a POS tag
#returns a two element array. First element is the text.
#Second is the POS tag
def tag(text):
    return pos_tag(text)

#Convert the array of tokens into one string per data point
def tokens_to_string(text):
    string = ""
    for i in text:
        string += " " + i
```

```
    return string


#Reads the training and testing sets into a pandas dataframe
test = pd.read_csv('drugLibTest_raw.tsv',delimiter='\t')
train = pd.read_csv('drugLibTrain_raw.tsv', delimiter='\t')


#Adds the review column with no values
#concatenates all the review columsn into the review column
#Removes redundant columsn such as the benefitsReview, etc. columns
train.insert(1, column="review", value=None)
train["review"] = train["benefitsReview"] + train["sideEffectsReview"]  + train["commentsReview"]
train = train.drop(["Unnamed: 0", "benefitsReview", "sideEffectsReview", "commentsReview", "urlDrugName", "condition"], axis=1)
test["review"] = test["benefitsReview"] + test["sideEffectsReview"]  + test["commentsReview"]
test = test.drop(["Unnamed: 0", "benefitsReview", "sideEffectsReview", "commentsReview", "urlDrugName", "condition"], axis=1)


#Tokenization
#Converts the text into a list of tokens and removes any punctuation
tokenizer = RegexpTokenizer(r'\w+')
train["review"] = train["review"].apply(lambda x: tokenizer.tokenize((str)(x).lower()))
test["review"] = test["review"].apply(lambda x: tokenizer.tokenize((str)(x).lower()))


# Remove stopwords
train["review"] = train["review"].apply(lambda x: remove_stop_words(x))
test["review"] = test["review"].apply(lambda x: remove_stop_words(x))


#POS tagging
train["review"] = train["review"].apply(lambda x: tag(x))
test["review"] = test["review"].apply(lambda x: tag(x))


# Lemmatization
train["review"] = train["review"].apply(lambda x: word_lemmatizer(x))
```

```
test["review"] = test["review"].apply(lambda x: word_lemmatizer(x))

#uncomment to stem the data
# Stemmerization
# train["review"] = train["review"].apply(lambda x: word_stemmer(x))
# test["review"] = test["review"].apply(lambda x: word_stemmer(x))

#convert the array of tokens to string
train["review"] = train["review"].apply(lambda x: tokens_to_string(x))
test["review"] = test["review"].apply(lambda x: tokens_to_string(x))

#convert the side effects into an int
train["sideEffects"] = train["sideEffects"].apply(lambda x: convert_side_effect_index(x))
test["sideEffects"] = test["sideEffects"].apply(lambda x: convert_side_effect_index(x))

#convert the effectiveness into an int
train["effectiveness"] = train["effectiveness"].apply(lambda x: convert_effectiveness_index(x))
test["effectiveness"] = test["effectiveness"].apply(lambda x: convert_effectiveness_index(x))

# Vectorization
# The vocab is restricted to 150 words
from sklearn.feature_extraction.text import  TfidfVectorizer
vectorizer=TfidfVectorizer(use_idf=True, max_features=150, ngram_range=(3, 3))

#convert the review column from the trainign data into a array of floats
x = vectorizer.fit_transform(train["review"]).toarray()
x_train = pd.DataFrame(x,columns=vectorizer.get_feature_names())
#Add the rating and effectiveness columns
x_train["rating"] = train["rating"]
x_train["effectiveness"] = train["effectiveness"]

x = vectorizer.fit_transform(test["review"]).toarray()
```

```
x_test = pd.DataFrame(x,columns=vectorizer.get_feature_names())
x_test["rating"] = test["rating"]
x_test["effectiveness"] = test["effectiveness"]

# removes invalid data that is such as NaN, Infinity, etc.
x_train = x_train.dropna()
x_test = x_test.dropna()

# Add the target columns to test and train
y_train = train["sideEffects"]
y_test = test["sideEffects"]

# Classifcation
from sklearn.metrics import classification_report
print("SVM")
from sklearn.svm import SVC
classifier = SVC(gamma='auto', C=1, class_weight='balanced')
classifier.fit(x_train, y_train)

#Prints out the classifcation report with the precision, recall, f-values and accuracy score
target_names = ['No Side Effects', 'Mild Side Effects', 'Moderate Side Effects', 'Severe Side Effects', 'Extremely Severe Side Effects']
y_pred = classifier.predict(x_test)
print(classification_report(y_test,y_pred, target_names=target_names))

print("Decision Tree")
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(max_depth=6, random_state=1)
classifier.fit(x_train, y_train)

#Prints out the classifcation report with the precision, recall, f-values and accuracy score
y_pred = classifier.predict(x_test)
print(classification_report(y_test,y_pred, target_names=target_names))
```

```
#Uncomment if you wish to run the grid search

#Grid search to find the optimal parameters

# from sklearn.model_selection  import GridSearchCV

# param_grid = {'C':[1,10,100,1000], 'gamma':[1,0.1,0.001,0.0001,'auto'], 'kernel':['linear','rbf'],'class_weight' : ['balanced', None]}

# grid = GridSearchCV(SVC(),param_grid,refit = True, verbose=2)

# grid.fit(x_train,y_train)

# print(grid.best_params_)


# param_grid = {'max_depth': np.arange(3, 10), "random_state" :[0, 1, None]}

# grid = GridSearchCV(DecisionTreeClassifier(), param_grid)

# grid.fit(x_train,y_train)

# print(grid.best_params_)
```