**Abstract**

**Introduction**

**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 tab separated columns format (.tsv).

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 |

**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. 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 and lemmatization. The second results are the results with rating and effectiveness added as features. 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. Two classifiers were used. The classifiers’ goal is to determine the severity of the side effects. As in give a classification of No Side Effects, Mild Side Effects, Severe Side Effects and Extremely Severe Side Effects. Support Vector Classification (SVC) classifier and Decision Tree classifier. The weighted average for the precision scores, recall score, f values and accuracy scores are collected (Macro average values are in Appendix X). 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 is the weighted average of recall and precision score. Accuracy Score is the ratio of correct results. The basic methodology for the text mining and the NLP 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. All text is also case folded by converting 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 its 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 human readable. 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 fairly aggressive, and is the highest performing stemming algorithm. (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 a 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 instead of just tracking the term frequency, contains a TFIDF value. TFIDF (Term frequency inverse document frequency), 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. When both of these values are combined they can be 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 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 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 sclassification.

**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.

**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 being tested. In regard to 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 coefficient I used was 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=1, gamma=auto, kernel=rbf and class weights=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, max depths=6 and random state=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** |

As we see in table 1 the SVM performs much better than the Decision Tree classifier. We see that theaverageprecisionscore for the SVMis 0.28 compare the Decision Tree which has a average precision score of 0.20. They both have a 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. These results indicate that the SVM model performed better than the Decision tree model.

**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** |

As we can see in table 2. The addition of effectiveness and rating as feature 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.

**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** |

**­­­**Shown in table 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 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.

**Discussion**

My results show the impact of various techniques in a effect to improve 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.

The second set of results also 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. 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 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 isn’t much data per drug for it to 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. I also performed hyperparameter optimization. I used a grid search as describe earlier. However, since of hardware and time constraint. If there was more time, then grid searching more parameters may lead to even more improvements.

For the third results the removal of the stemming process and adding POS improved the results, and unlike the previous results the Decision tree, performed better than the SVM. A reason that possibly explain the better performance of stemming is that the Porter’s Algorithm used for stemming might be too aggressive. This means that sometimes the Porter stemmer may process words into the same word stem, but the words may have different semantic difference. 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 experimentation in-order to determine if other stemmers are effective. The POS tagging also helped slightly, for the Decision Tree Classifier. However, there was little experimentation with different POS tagging methods. The Lemmatiz

**Conclusion**

# 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. Just incase someone wants to run those for testing purposes

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

def remove\_stop\_words(text):

words = [w for w in text if w not in stopwords]

return words

lemmatizer = WordNetLemmatizer()

def word\_lemmatizer(text):

lem\_text = []

for i in text:

pos = i[1]

pos = convert\_tag(pos)

x = None

if pos == '':

x = lemmatizer.lemmatize(i[0])

else:

x = lemmatizer.lemmatize(i[0], pos=pos)

lem\_text.append(x)

return lem\_text

stemmer = PorterStemmer()

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 ''

def word\_stemmer(text):

stem\_text = []

for i in text:

x = stemmer.stem(i)

stem\_text.append(x)

return stem\_text

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

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

def tag(text):

return pos\_tag(text)

def tokens\_to\_string(text):

string = ""

for i in text:

string += " " + i

return string

test = pd.read\_csv('drugLibTest\_raw.tsv',delimiter='\t') # Read the files with the pandas dataFrame

train = pd.read\_csv('drugLibTrain\_raw.tsv', delimiter='\t')

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))

train["review"] = train["review"].apply(lambda x: tag(x))

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

print("lemmatization")

# Lemmatization

train["review"] = train["review"].apply(lambda x: word\_lemmatizer(x))

test["review"] = test["review"].apply(lambda x: word\_lemmatizer(x))

# Stemmerization

# train["review"] = train["review"].apply(lambda x: word\_stemmer(x))

# test["review"] = test["review"].apply(lambda x: word\_stemmer(x))

# 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))

train["sideEffects"] = train["sideEffects"].apply(lambda x: convert\_side\_effect\_index(x))

test["sideEffects"] = test["sideEffects"].apply(lambda x: convert\_side\_effect\_index(x))

train["effectiveness"] = train["effectiveness"].apply(lambda x: convert\_effectiveness\_index(x))

test["effectiveness"] = test["effectiveness"].apply(lambda x: convert\_effectiveness\_index(x))

from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer, TfidfTransformer

vectorizer=TfidfVectorizer(use\_idf=True, max\_features=150, ngram\_range=(3, 3))

x = vectorizer.fit\_transform(train["review"]).toarray()

x\_train = pd.DataFrame(x,columns=vectorizer.get\_feature\_names())

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"]

x\_train = x\_train.dropna()

x\_test = x\_test.dropna()

y\_train = train["sideEffects"]

y\_test = test["sideEffects"]

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)

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)

y\_pred = classifier.predict(x\_test)

print(classification\_report(y\_test,y\_pred, target\_names=target\_names))

# 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\_)