Final Term Project

Submission Details

By: **Parikshit Narang**

NJIT ID: **31530064**

Course: **CS634 - Data Mining**

New Jersey Institute of Technology

Contents

Introduction

Sentiment Analysis, also known as Opinion Mining is the process of analysing various stakeholder or customer reviews to detect positive or negative sentiment. It is one of the important tools used by businesses to get to know about their brand reputation and understand customer's varied needs.

This project deals with analysis of sentiments by making use of three different classifiers: Support Vector Machine (SVM), Random Forest and Naive Bayes. As we have reviews labelled as, positive and negative, for textual reviews we can use these machine learning models for analysis. Also, we have the raw text, we will first preprocess the data using NLTK and extract certain features using Tf-IDF vectorizer.

The subsequent sections will describe the problem definition and algorithm, experimental evaluation and future work.

Problem Statement

The goal is to create a sentiment analyser using different classification algorithms. The tasks involved are the following:

1. Download and preprocess the Sentiment Labelled Sentences data set from UCI Machine Learning Repository
2. Train three different classifiers using the preprocessed data
3. Use K-fold cross validation on the training data for every classifier
4. Generate report comprising of different scores like specificity, sensitivity, TSS, HSS, etc. and confusion matrices for every iteration of K-Fold cross validation
5. Compare the accuracy scores of customised K-Fold Cross Validation and SKLearn's K-Fold Cross Validation
6. Predict the sentiments using the trained models
7. Use the predictions to generate the confusion matrix for every model
8. Generate report using the confusion matrix obtained comprising of different scores like specificity, sensitivity, TSS, HSS, etc. and confusion matrices for every model

Metrics

The metrics taken into account while comparing the model performance are as follows:

1. ***Confusion Matrix:*** It is known as "error matrix" and can be used for either binary or multi classification problems. It evaluates the model performance by evaluating true positive, false positive, false negative and true negative.

2. ***Recall or Sensitivity:*** It is the ratio of no. of data points classified correctly as positive to no. of data points that are actually positive.

3. ***Specificity:*** It is the ratio of no. of data points classified correctly as negative to no. of data points classified as negative.

4. ***Precision:*** It is the ratio of no. of data points classified correctly as positive to no. of data points classified as positive.

5. ***Negative Predictive Value:*** It is the ratio of no. of data points classified correctly as negative to no. of data points classified as negative.

6. ***False Positive Rate:*** It is the ratio of no. of data points classified correctly as positive to no. of data points that are actually negative.

7. **False Discovery Rate:** It is the ratio of no. of data points wrongly classified as positive to the no. of data points classified as positive.

8. ***False Negative Rate:*** It is the ratio of no. of data points wrongly classified as negative to the no. of data points that are actually positive.

9. ***Accuracy:*** It is the ratio of no. of data points classified correctly to the total no. of data points.

10. ***F1 Score:*** It is defined as the harmonic mean of model's precision and recall.

11. ***Balanced Accuracy:*** It measures the average sensitivity and specificity.

12. ***True Skill Statistics:*** It measures the difference between recall minus the probability of false detection.

13. ***Heidke Skill Score:*** It measures the fractional prediction over random prediction.

14. ***Brier Score:*** It is the mean squared error between expected probabilities and the predicted probabilities.

15. ***Brier Skill Score:*** It measure the likelihood of an event to happen.

16. ***K-fold cross validation:*** It is the resampling procedure used to evaluate the machine learning models on limited data sample. It has a single parameter k which refers to the number of groups data sample is to be split into.

Methodology

This section describes the methodology followed while implementing the sentiment analysis using NLP.

Data Extraction

For extracting data, dataset is downloaded, saved locally and then fetched in dataframe through pandas library. The extracted data contain reviews from three different websites: imdb.com, amazon.com and yelp.com. Figure 1.1 depicts the amazon data in dataframe, figure 1.2 depicts the imdb data in dataframe and figure 1.3 depicts the yelp data in dataframe. Refer appendix A.1 for data extraction.

	0	1
0	So there is no way for me to plug it in here i...	0
1	Good case, Excellent value.	1
2	Great for the jawbone.	1
3	Tied to charger for conversations lasting more...	0
4	The mic is great.	1
...	...	...
995	The screen does get smudged easily because it ...	0
996	What a piece of junk.. I lose more calls on th...	0
997	Item Does Not Match Picture.	0
998	The only thing that disappoint me is the infra...	0
999	You can not answer calls with the unit, never ...	0

1000 rows × 2 columns

Figure 1.1: Amazon Data

	0	1
0	A very, very, very slow-moving, aimless movie ...	0
1	Not sure who was more lost - the flat characte...	0
2	Attempting artiness with black & white and cle...	0
3	Very little music or anything to speak of.	0
4	The best scene in the movie was when Gerardo i...	1
...	...	...
743	I just got bored watching Jessice Lange take h...	0
744	Unfortunately, any virtue in this film's produ...	0
745	In a word, it is embarrassing.	0
746	Exceptionally bad!	0
747	All in all its an insult to one's intelligence...	0

748 rows × 2 columns

Figure 1.2: IMDB Data

	0	1
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1
...	...	...
995	I think food should have flavor and texture an...	0
996	Appetite instantly gone.	0
997	Overall I was not impressed and would not go b...	0
998	The whole experience was underwhelming, and I ...	0
999	Then, as if I hadn't wasted enough of my life ...	0

1000 rows × 2 columns

Figure 1.3: Yelp Data

Data Preprocessing

The data fetched is in raw format or unprocessed state. For preprocessing of data, we encode the sentiment values, 0 as 'negative' and 1 as 'positive'. Refer appendix B.1 for encoding.

We, then remove all the stop words, occurring in NLTK corpus, from the fetched data. Refer appendix B.2 for stop words removal.

Finally, we extract the features from training data using TfIdf-vectorizer. We randomly select features as 30 and these features also comprise of one word sequence (1-gram) and two words sequence (2-gram/ bi-gram). Refer appendix B.3 for features extraction using TfIDF vectorizer.
Figure 1.4 depicts the preprocessed data in dataframe.

	also	back	bad	best	could	even	ever	film	food	go	...	place	product	quality	really	service	the	time	ve	well	would
0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
1	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	1.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
2	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.702693	0.0	0.0	0.000000	0.0	0.0	0.711493	0.0	0.0	0.0
3	0.000000	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
4	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1836	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
1837	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
1838	0.784914	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
1839	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.730721	0.0	0.0	0.000000	0.0	0.0	0.0
1840	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.0	0.000000	0.0	0.0	1.000000	0.0	0.0	0.0

1841 rows × 30 columns

Figure 1.4: Preprocessed Data

Model Training

The preprocessed data is now fed to three different classifiers: Support Vector Machine(SVM), Random Forest and Naive Bayes along with appropriately hyper-tuned parameters. Refer appendix C for model training.

Model Evaluation

The trained model/classifier is then evaluated on the basis of K-Fold Cross Validation and metrics discussed in one of the previous sections.

For K-Fold Cross Validation, we build our own K-Fold Cross Validation function and then compare it with the SKLearn's Cross Validation to verify if it's correct. Refer appendix D.1 for custom K-Fold Cross Validation function. Also, we take K=10. The customised K-fold cross validation takes argument as classifier, x_train, y_train, no of folds and shuffle value as input. It divides the x_train into (K-1) sets. The (K-1) sets indicates training data and K^{th} set indicates testing data. The model is trained and tested using the corresponding training and testing data. This is performed K number of times and accuracy scores are noted down for every iteration of K-Fold Cross Validation. Finally, we get the mean of accuracy scores. This gives us an idea about the performance of model/classifier on unseen dataset.

For each iteration of K-Fold Cross Validation, we generate confusion matrix and on the basis of confusion matrix we calculate different scores like sensitivity, specificity, TSS, HSS, and so on.

Model Comparison

The previous section discusses about individual model's/classifier's comparison against different metrics. Now, we compare the models with each other against the metrics stated. Refer appendix E.1 for model comparison function.

Result Analysis

This section describes the comparison analysis of models against different metrics. The various scores against which we evaluate the model forms the basis for result analysis. Figure 1.5 depicts the model performance comparison against different evaluation metrics. Figure 1.6 depicts the confusion matrix for SVM, Random Forest and Naive Bayes.

Measure	SVM	Random Forest	Naive Bayes
Sensitivity	0.6990740740740741	0.9236111111111112	0.75
Specificity	0.5157894736842106	0.16210526315789472	0.4357894736842105
Precision	0.5676691729323309	0.5006273525721455	0.5472972972972973
Negative Predictive Value	0.6533333333333333	0.7	0.6571428571428571
False Positive Rate	0.4842105263157895	0.8378947368421052	0.5642105263157895
False Discovery Rate	0.4323308270676692	0.4993726474278545	0.4527027027027027
False Negative Rate	0.30092592592592593	0.0763888888888889	0.25
Accuracy	0.2971211298207496	0.2585551330798479	0.2884302009777295
F1 Score	0.6265560165975104	0.6493083807973963	0.6328125
BACC	0.6074317738791424	0.542858187134503	0.5928947368421053
TSS	0.2148635477582846	0.08571637426900591	0.1857894736842105
HSS	0.2126356402218471	0.08255905373214893	0.18272622699386504

Figure 1.5: Model Performance Comparison

```
SVM Confusion Matrix =>
+----------------+-------+
| TN, FP, FN, TP | Count |
+----------------+-------+
| True Negative  |  245  |
| False Postive  |  230  |
| False Negative |  130  |
|  True Postive  |  302  |
+----------------+-------+

Random Forest Confusion Matrix =>
+----------------+-------+
| TN, FP, FN, TP | Count |
+----------------+-------+
| True Negative  |   77  |
| False Postive  |  398  |
| False Negative |   33  |
|  True Postive  |  399  |
+----------------+-------+

Naive Bayes Confusion Matrix =>
+----------------+-------+
| TN, FP, FN, TP | Count |
+----------------+-------+
| True Negative  |  207  |
| False Postive  |  268  |
| False Negative |  108  |
|  True Postive  |  324  |
+----------------+-------+
```

Figure 1.6: Confusion Matrices for SVM, Random Forest and Naive Bayes

---

Result Basis

- ***Sensitivity*****:** Higher sensitivity means that we have few false negative results. Thus, we can conclude Random Forest performs better as compared to SVM and Naive Bayes. Random Forest ~ 92%, Naive Bayes ~ 75% and SVM ~ 70%.

- ***Specificity*****:** Higher sensitivity means that we have few false positive results. Thus, we can conclude SVM performs better as compared to Naive Bayes and Random Forest. SVM ~ 52%, Naive Bayes ~ 44% and Random Forest ~ 16%.

- ***Precision*****:** Higher precision gives us the measure of relevant data points. Thus, we can conclude SVM performs better as compared to Naive Bayes and Random Forest. SVM ~ 57%, Naive Bayes ~ 55% and Random Forest ~ 51%.

- ***Negative Predictive Value*****:** Higher negative predictive value means high true negative results out of total negatives. Thus, we conclude Random Forest performs better as compared to Naive Bayes and SVM. Random Forest ~ 70%, Naive Bayes ~ 66% and SVM ~ 65%.

- ***False Positive Rate*****:** Higher false positive rate means that the model will predict wrong value that is false positive given the training data point. Thus, we can conclude, SVM performs better as compared to Naive Bayes and Random Forest. SVM ~ 48%, Naive Bayes ~ 56% and Random Forest ~ 84%.

- ***False Discovery Rate*:** Higher false discovery rate means that the model will predict wrong value that is false positive given a training data point. Thus, we can conclude, SVM performs better as compared to Naive Bayes and Random Forest. SVM ~ 43%, Naive Bayes ~ 45% and Random Forest ~ 50%.

- ***False Negative Rate*:** Higher false positive rate means that the model will predict wrong value that is false negative given the training data point. Thus, we can conclude, Random Forest performs better as compared to Naive Bayes and SVM. Random Forest ~ 7%, Naive Bayes ~ 25% and SVM ~ 30%.

- ***Accuracy*:** High accuracy means higher the number of correct predictions. Thus, we can conclude SVM is much better as compared to Naive Bayes and Random Forest. SVM ~ 30%, Naive Bayes ~ 29% and Random Forest 26%.

- ***F1 Score*:** High F1 Score means good precision and recall. Thus, we can conclude Random Forest performs better as compared to Naive Bayes and SVM. Random Forest ~ 65%, Naive Bayes ~ 63% and SVM ~ 62%.

- ***BACC*:** High BACC means good sensitivity and specificity on an average. Thus, we can conclude SVM performs better as compared to Naive Bayes and Random Forest. SVM ~ 61%, Naive Bayes ~ 59% and Random Forest ~ 54%.

- ***TSS*:** Higher TSS means more sensitivity and less false positive rate. Thus, we can conclude SVM performs better as compared to Naive Bayes and Random Forest. SVM ~ 21%, Naive Bayes ~ 19% and Random Forest ~ 9%.

- ***HSS*:** Higher HSS means more fractional improvement in prediction over standard/random prediction. Thus, we can conclude SVM performs better as compared to Naive Bayes and Random Forest. SVM ~ 21%, Naive Bayes ~ 18% and Random Forest ~ 8%.

Appendices

Appendix A

Data Fetching

A.1 Sentiment Dataset Extraction

```
import pandas as pd
from nltk import pos_tag
from nltk.tokenize import word_tokenize

amazon_data =  pd.read_csv('amazon_cells_labelled.txt', sep="\t", header=None)
imdb_data = pd.read_csv('imdb_labelled.txt', sep="\t", header=None)
yelp_data = pd.read_csv('yelp_labelled.txt', sep="\t", header=None)
```

Appendix B

Data Preprocessing

B.1 Encoding

```
data = pd.DataFrame()
data = data.append(amazon_data)
data = data.append(imdb_data)
data = data.append(yelp_data)
data.columns = ['sentence', 'sentiment']

def preprocess_sentiment(x):
    if x==0:
        return 'negative'
    else :
        return 'positive'

data['sentiment']=data['sentiment'].apply(preprocess_sentiment)
```

B.2 Stop Words Removal

```
from nltk.corpus import stopwords
stop = stopwords.words('english')

from sklearn.model_selection import train_test_split
x_train_intermediate, x_test_intermediate, y_train, y_test = train_test_split(data['sentence'],
data['sentiment'], test_size=0.33, random_state=42)

x_train_splitted = [xTrainIntermediate.split(" ") for xTrainIntermediate in x_train_intermediate]
x_train_splitted

x_train_list = [[xTrain] for xTrain in x_train_intermediate]

index = 0
for sentence in x_train_list:
    for word in sentence[0].split(" "):
        if word.lower() in stop:
            x_train_splitted[index].remove(word)
    index = index + 1
```

B.3 Feature Extraction

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_features = 30, ngram_range=(1, 2))
vectors = vectorizer.fit_transform(x_train)

vectorizer.get_feature_names()
```

Appendix C

Model Training

C.1 Support Vector Machine (SVM)

```
from sklearn.svm import SVC
svc = SVC(C=1.0, kernel='rbf', coef0=1, gamma=1)

svc.fit(x_train, y_train)
x_test = vectorizer.transform(x_test_intermediate).todense()
y_pred = svc.predict(x_test)
```

C.2 Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(max_depth=2, random_state=0)
rfc.fit(x_train, y_train)

y_pred = rfc.predict(x_test)
```

C.3 Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)

y_pred = gnb.predict(x_test)
```

Appendix D

K-Fold Cross Validation

D.1 Custom K-Fold Cross Validation Function

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
from prettytable import PrettyTable
from sklearn.utils import shuffle

def kfoldCrossValidation(clf, x_train, y_train, noOfFolds, shuffleValue):
    kfold_df_x = x_train
    kfold_df_x = kfold_df_x.reset_index(drop=True)

    kfold_df_y = y_train
    kfold_df_y = kfold_df_y.reset_index(drop=True)

    kfold_df = kfold_df_x.join(kfold_df_y)

    if shuffleValue==True:
        kfold_df = shuffle(kfold_df)
    list_df = []
    temp = 0
    for i in range(0,noOfFolds,1):
        first_slicing_value = temp;
        second_slicing_value = first_slicing_value +  (len(kfold_df) + 1) //noOfFolds
        list_df.append(kfold_df[first_slicing_value : second_slicing_value])
        temp = second_slicing_value + 1

    accuracy_scores = []
    for i in range(0, len(list_df)):
        print("\nIteration ",(i+1)," for K-Fold Cross Validation : ")
        print("==================================================")
        x_test_cv = list_df[i].iloc[:, :list_df[i].shape[1]-1]
        y_test_cv = list_df[i].iloc[:, list_df[i].shape[1]-1 : list_df[i].shape[1]+1]

        x_train_cv = pd.DataFrame()
        y_train_cv = pd.DataFrame()
        for j in range(0, len(list_df)):
            if j!=i:
                x_train_cv = x_train_cv.append(list_df[j])

        y_train_cv = x_train_cv.iloc[:, x_train_cv.shape[1]-1 : x_train_cv.shape[1]+1]
        x_train_cv = x_train_cv.iloc[:, :x_train_cv.shape[1]-1]
        print(x_train_cv)

        clf.fit(x_train_cv, y_train_cv)
        y_test_cv_pred = clf.predict(x_test_cv)

        results = metrics.accuracy_score(y_test_cv, y_test_cv_pred)
        print("Accuracy Score : ",results)
        accuracy_scores.append(results)

        labels  = np.unique(np.array(y_test_cv))
        cm = confusion_matrix(np.array(y_test_cv), np.array(y_test_cv_pred), labels = labels)
        print("\n\nConfusion Matrix ==>\n")
        print(pd.DataFrame(cm, index=labels, columns=labels))

        generate_reports(cm)

    print("Mean Accuracy Score : ",np.array(accuracy_scores).mean())
```

Appendix E

Model Comparison

E.1 Generate Comparison

```
def generate_comparison(cm1, cm2, cm3):
        tn1, fp1, fn1, tp1 = cm1.ravel()
        table = PrettyTable()
        table.field_names = ["TN, FP, FN, TP", "Count"]
        table.add_row(["True Negative", tn1])
        table.add_row(["False Postive", fp1])
        table.add_row(["False Negative", fn1])
        table.add_row(["True Postive", tp1])

        print("\nSVM Confusion Matrix =>")
        print(table)

        tn2, fp2, fn2, tp2 = cm2.ravel()
        table = PrettyTable()
        table.field_names = ["TN, FP, FN, TP", "Count"]
        table.add_row(["True Negative", tn2])
        table.add_row(["False Postive", fp2])
        table.add_row(["False Negative", fn2])
        table.add_row(["True Postive", tp2])

        print("\nRandom Forest Confusion Matrix =>")
        print(table)

        tn3, fp3, fn3, tp3 = cm3.ravel()
        table = PrettyTable()
        table.field_names = ["TN, FP, FN, TP", "Count"]
        table.add_row(["True Negative", tn3])
        table.add_row(["False Postive", fp3])
        table.add_row(["False Negative", fn3])
        table.add_row(["True Postive", tp3])

        print("\nNaive Bayes Confusion Matrix =>")
        print(table)

        sen1 = tp1 / (tp1 + fn1)
        sen2 = tp2 / (tp2 + fn2)
        sen3 = tp3 / (tp3 + fn3)
        # print("Sensitivity : ", sen)

        spec1 = tn1 / (fp1 + tn1)
        spec2 = tn2 / (fp2 + tn2)
        spec3 = tn3 / (fp3 + tn3)
        # print("Specificity : ", spec)

        prec1 = tp1 / (tp1 + fp1)
        prec2 = tp2 / (tp2 + fp2)
        prec3 = tp3 / (tp3 + fp3)
        # print("Precision : ", prec)

        npv1 = tn1 / (tn1 + fn1)
        npv2 = tn2 / (tn2 + fn2)
        npv3 = tn3 / (tn3 + fn3)
        # print("Negative Predictive Value : ", npv)
```

```
fpr1 = fp1 / (fp1 + tn1)
fpr2 = fp2 / (fp2 + tn2)
fpr3 = fp3 / (fp3 + tn3)
# print("False Positive Rate : ", fpr)

fdr1 = fp1 / (fp1 + tp1)
fdr2 = fp2 / (fp2 + tp2)
fdr3 = fp3 / (fp3 + tp3)
# print("False Discovery Rate : ", fdr)

fnr1 = fn1 / (fn1 + tp1)
fnr2 = fn2 / (fn2 + tp2)
fnr3 = fn3 / (fn3 + tp3)
# print("False Negative Rate : ", fnr)

acc1 = (tp1 + tn1) / (len(y_train))
acc2 = (tp2 + tn2) / (len(y_train))
acc3 = (tp3 + tn3) / (len(y_train))
# print("Accuracy : ", acc)

f1score1 = (2 * tp1) / (2*tp1 + fp1 + fn1)
f1score2 = (2 * tp2) / (2*tp2 + fp2 + fn2)
f1score3 = (2 * tp3) / (2*tp3 + fp3 + fn3)
# print("F1 Score : ", f1score)

bacc1 = 0.5 * (sen1 + spec1)
bacc2 = 0.5 * (sen2 + spec2)
bacc3 = 0.5 * (sen3 + spec3)
# print("BACC : ", bacc)

tss1 = sen1 - fpr1
tss2 = sen2 - fpr2
tss3 = sen3 - fpr3
# print("TSS : ", tss)

hss1 = (2 * (tp1 * tn1 - fp1 * fn1)) / (((tp1 + fn1)* (fn1 + tn1)) + (tp1 + fp1) * (fp1 +
       tn1))
hss2 = (2 * (tp2 * tn2 - fp2 * fn2)) / (((tp2 + fn2)* (fn2 + tn2)) + (tp2 + fp2) * (fp2 +
       tn2))
hss3 = (2 * (tp3 * tn3 - fp3 * fn3)) / (((tp3 + fn3)* (fn3 + tn3)) + (tp3 + fp3) * (fp3 +
       tn3))
# print("HSS : ", hss)

table = PrettyTable()
table.field_names = ["Measure", "SVM", "Random Forest", "Naive Bayes"]
table.add_row(["Sensitivity", sen1, sen2, sen3])
table.add_row(["Specificity", spec1, spec2, spec3])
table.add_row(["Precision", prec1, prec2, prec3])
table.add_row(["Negative Predictive Value", npv1, npv2, npv3])
table.add_row(["False Positive Rate", fpr1, fpr2, fpr3])
table.add_row(["False Discovery Rate", fdr1, fdr2, fdr3])
table.add_row(["False Negative Rate", fnr1, fnr2, fnr3])
table.add_row(["Accuracy", acc1, acc2, acc3])
table.add_row(["F1 Score", f1score1, f1score2, f1score3])
table.add_row(["BACC", bacc1, bacc2, bacc3])
table.add_row(["TSS", tss1, tss2, tss3])
table.add_row(["HSS", hss1, hss2, hss3])

print(table)
```