Brain Tumor Detection by Using K-Nearest Neighbor, Support Vector Machine and Naïve Bayes Classifiers

Ismet Okan Celik

CWID: 10472265

Stevens Institute of Technology

CPE 646 - Pattern Recognition and Classification

Abstract

According to the American Cancer Society, 13,840 in males and 10,690 in females will be diagnosed with brain or spinal cord tumors in 2021. Approximately 10,500 males and 8,100 females will die due to the brain tumor. Brain tumors can be grown during the time and can be more complex. For avoiding a fatal situation, it is vital to diagnose the brain tumors as early as possible to increase the survival rate for people with brain tumors. One of the best ways to diagnose a brain tumor is to examine Magnetic Resonance Imaging (MRI) which should analyze by a professional neurosurgeon. Considering the developing countries, it is challenging and time-consuming to find a professional neurosurgeon or skillful doctor who can analyze MRI images. Brain tumors have complex nature considering their sizes and locations. It is even hard to diagnose manually for professionals. Manual diagnoses can be error-prone. For these reasons, using Machine Learning and Artificial Intelligence techniques can be more consistent and show higher accuracy than manual analysis. This project used K-Nearest Neighbor, Support Vector Machine, and Naïve Bayes classification model to detect brain tumors by using Kaggle Brain Tumor Dataset. Principal Component Analysis (PCA) was applied to reduce the dimension and avoid feature loss. After that, three different Machine Learning approaches were applied, and results were compared.

Keywords: K-Nearest Neighbor, Support Vector Machine, Naïve Bayes, Classification, Brain Tumor

1. Objective of the Project

The purpose of the project is to implement K-Nearest Neighborhood, Support Vector Machine, and Naïve Bayes classification method to solve brain tumor classification problem and determine the best performs algorithm by using the “Brain Tumor” dataset provided by Jakesh Bohaju, Kaggle, 2020.

2. Tools and Approach

The data set includes ten features (Mean, Variance, Standard Deviation, Entropy, Skewness, Kurtosis, Contrast, Energy, ASM (Angular Second Moment), Homogeneity, Dissimilarity, Correlation, Coarseness), and MRI images are labeled as Tumor=1 and Non-Tumor=0 under Class.

2.1. Data Importing

First, the data downloaded from: https://www.kaggle.com/jakeshbohaju/brain-tumor/download, after data imported by using pandas.

```
[266]: dataset=pd.read_csv('Brain Tumor.csv')
       dataset
```

[266]:

	Image	Class	Mean	Variance	Standard Deviation	Entropy	Skewness	Kurtosis	Contrast	Energy	ASM	Homogeneity	Dissimilarity	Correlation	Coarseness
0	Image1	0	6.535339	619.587845	24.891522	0.109059	4.276477	18.900575	98.613971	0.293314	0.086033	0.530941	4.473346	0.981939	7.458341e-155
1	Image2	0	8.749969	805.957634	28.389393	0.266538	3.718116	14.464618	63.858816	0.475051	0.225674	0.651352	3.220072	0.988834	7.458341e-155
2	Image3	1	7.341095	1143.808219	33.820234	0.001467	5.061750	26.479563	81.867206	0.031917	0.001019	0.268275	5.981800	0.978014	7.458341e-155
3	Image4	1	5.958145	959.711985	30.979219	0.001477	5.677977	33.428845	151.229741	0.032024	0.001026	0.243851	7.700919	0.964189	7.458341e-155
4	Image5	0	7.315231	729.540579	27.010009	0.146761	4.283221	19.079108	174.988756	0.343849	0.118232	0.501140	6.834689	0.972789	7.458341e-155
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3757	Image3758	0	21.234512	1208.850174	34.768523	0.063774	2.082079	4.647310	158.437600	0.220666	0.048693	0.487131	5.211739	0.950972	7.458341e-155
3758	Image3759	0	20.435349	1227.151440	35.030721	0.066763	2.144625	4.882034	161.158675	0.225931	0.051045	0.502712	5.083126	0.952749	7.458341e-155
3759	Image3760	0	18.011520	1151.582765	33.934978	0.068396	2.308349	5.579498	167.130118	0.228930	0.052409	0.492269	5.103700	0.952181	7.458341e-155
3760	Image3761	0	13.330429	945.732779	30.752769	0.087872	2.732822	7.757570	223.812932	0.261527	0.068397	0.480064	6.439784	0.940898	7.458341e-155
3761	Image3762	0	6.110138	480.884025	21.929068	0.118171	4.110669	17.538826	239.251388	0.306224	0.093773	0.494333	6.787329	0.938731	7.458341e-155

3762 rows × 15 columns

Fig. 1. Data Framework

Fig. 2. shows that the white part on the MRI images represents the tumors on the brain. The Features are a numerical representation of these MRI images that are going to use for classification.

```
[155]: plt.figure(figsize=(20,20))
       folder='Brain Tumor'
       for i in range(5):
           file=random.choice(os.listdir(folder))
           image_path=os.path.join(folder,file)
           img=mpimg.imread(image_path)
           ax=plt.subplot(1,5,i+1)
           ax.title.set_text(file)
           plt.imshow(img)
```

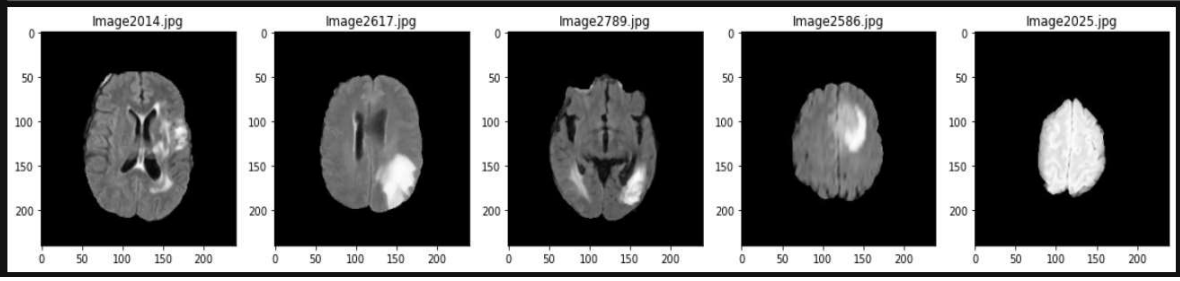


Fig. 2. Representation of Random Chosen MRI Images From Dataset

2.2. Data Scaling

Scaling is crucial before starting to classification, in Fig. 1. The numerical values of some of the features are much bigger than other features. For that reason, these features can dominate the other features, which have smaller values. Avoiding this situation and comparing all the features equally, the scaling process must be applied before classification for this project.

```
[9]: Features_Scaling=['Mean', 'Variance', 'Standard Deviation', 'Entropy','Skewness', 'Kurtosis', 'Contrast', 'Energy', 'ASM', 'Homogeneity','Dissimilarity',
                       'Correlation', 'Coarseness']
     dataset[Features_Scaling]=StandardScaler().fit_transform(dataset[Features_Scaling])
     dataset
```

[9]:

	Image	Class	Mean	Variance	Standard Deviation	Entropy	Skewness	Kurtosis	Contrast	Energy	ASM	Homogeneity	Dissimilarity	Correlation	Coarseness
0	Image1	0	-0.515700	-0.195790	-0.033144	0.504650	0.067855	-0.097267	-0.268050	0.685118	0.470073	0.404100	-0.121709	1.000713	-1.117857e-168
1	Image2	0	-0.129018	0.202943	0.365594	2.746050	-0.150204	-0.175881	-0.585492	2.090287	2.865580	1.345457	-0.799181	1.264377	-1.117857e-168
2	Image3	1	-0.375013	0.925765	0.984680	-1.026708	0.374531	0.037048	-0.421010	-1.335981	-0.988340	-1.649391	0.693704	0.850636	-1.117857e-168
3	Image4	1	-0.616481	0.531896	0.660820	-1.026561	0.615188	0.160202	0.212525	-1.335154	-0.988223	-1.840334	1.622995	0.322041	-1.117857e-168
4	Image5	0	-0.379529	0.039451	0.208352	1.041256	0.070489	-0.094103	0.429532	1.075849	1.022442	0.171115	1.154744	0.650854	-1.117857e-168
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3757	Image3758	0	2.050827	1.064921	1.092779	-0.139892	-0.789131	-0.349862	0.278359	0.123407	-0.170487	0.061596	0.277439	-0.183335	1.338116e-167
3758	Image3759	0	1.911290	1.104076	1.122668	-0.097345	-0.764704	-0.345702	0.303212	0.164116	-0.130150	0.183409	0.207915	-0.115377	1.338116e-167
3759	Image3760	0	1.488082	0.942399	0.997760	-0.074100	-0.700765	-0.333342	0.357754	0.187301	-0.106751	0.101766	0.219037	-0.137095	1.338116e-167
3760	Image3761	0	0.670746	0.501988	0.635006	0.203090	-0.534994	-0.294742	0.875475	0.439344	0.167517	0.006346	0.941273	-0.568517	1.338116e-167
3761	Image3762	0	-0.589942	-0.492543	-0.370847	0.634342	0.003102	-0.121400	1.016485	0.784933	0.602848	0.117900	1.129143	-0.651387	1.348053e-167

3762 rows × 15 columns

Fig. 3. Scaled Data Framework

2.3. Principal Component Analysis (PCA)

Making classifications is challenging if the dimension of the feature vector is significant. In this project, there are ten features and making calculations with this dimension is complex and complicated. For simplicity of the classification, the PCA method needs to be used in order to reduce the dimension of the feature vector while preserving as much information as possible.

```
[269]: # PCA Dimension Reduction
       data_prep=dataset.drop(['Image','Class'],axis=1,inplace=False)
       pca=PCA(n_components=2)
       pca_data=pca.fit_transform(data_prep)

[270]: pca_dataframe=pd.DataFrame(data=pca_data,columns=['PCA-1','PCA-2'])
```

Fig. 4. Principal Component Analysis (PCA)

```
[297]: df=pd.DataFrame(data)
       df['PCA-1']=pca_dataframe['PCA-1']
       df['PCA-2']=pca_dataframe['PCA-2']
       df.tail()
```

[297]:

	Image	Class	Mean	Variance	Standard Deviation	Entropy	Skewness	Kurtosis	Contrast	Energy	ASM	Homogeneity	Dissimilarity	Correlation	Coarseness	PCA-1	PCA-2
3757	Image3758	0	2.050827	1.064921	1.092779	-0.139892	-0.789131	-0.349862	0.278359	0.123407	-0.170487	0.061596	0.277439	-0.183335	1.338589e-167	0.321001	-2.161311
3758	Image3759	0	1.911290	1.104076	1.122668	-0.097345	-0.764704	-0.345702	0.303212	0.164116	-0.130150	0.183409	0.207915	-0.115377	1.338589e-167	0.205905	-2.131569
3759	Image3760	0	1.488082	0.942399	0.997760	-0.074100	-0.700765	-0.333342	0.357754	0.187301	-0.106751	0.101766	0.219037	-0.137095	1.338589e-167	0.203598	-1.775664
3760	Image3761	0	0.670746	0.501988	0.635006	0.203090	-0.534994	-0.294742	0.875475	0.439344	0.167517	0.006346	0.941273	-0.568517	1.338589e-167	0.273145	-0.655034
3761	Image3762	0	-0.589942	-0.492543	-0.370847	0.634342	0.003102	-0.121400	1.016485	0.784933	0.602848	0.117900	1.129143	-0.651387	1.348525e-167	-0.308959	1.213269

Fig. 5. Displaying Principal Component Analysis (PCA) on Data Framework

2.3. Data Splitting Operation

In order to evaluate the performance of the classification model, the dataset has to be divided into train dataset and test dataset. Train dataset used for training of the model, test dataset used for estimating the performance of the trained model. Common split percentages include:

- Train: 80%, Test: 20%
- Train: 67%, Test: 33%
- Train: 50%, Test: 50% (Brownlee, 2020)

In this project, 80 percent of data was used for training 20 percent of data used for testing.

```
[272]: #Data Spliting Operation
       X=df[['PCA-1','PCA-2']]
       X=np.array(X)
       Y=data['Class']
       Y=np.array(Y)

       # Splitting Data ----> 20 percent of data is used for testing, and 80 percent of data is used for trainning
       X_train,X_test, Y_train, Y_test=train_test_split(X,Y,test_size=0.2)
       print('X_train Shape =', X_train.shape)
       print('X_test Shape =', X_test.shape)
       print('Y_train Shape =', Y_train.shape)
       print('Y_test Shape =', Y_test.shape)

       X_train Shape = (3009, 2)
       X_test Shape = (753, 2)
       Y_train Shape = (3009,)
       Y_test Shape = (753,)
```

Fig. 6. Data Splitting Operation

```
[299]: #Training Data Frame With Class Labels
       x_train=pd.DataFrame(data=X_train,columns=['PCA-1','PCA-2'])
       x_train['Class']=pd.DataFrame(Y_train)

[274]: #Plotting Training Samples
       sns.lmplot('PCA-1','PCA-2',x_train,hue='Class',fit_reg=False)
       fig = plt.gcf()
       fig.set_size_inches(6, 3)
       plt.show()

C:\Users\okanc\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y, data. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

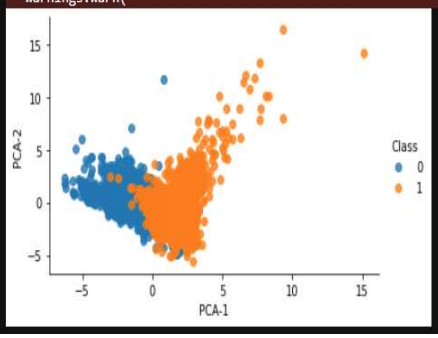


Fig. 7. Training Dataset (Tumor=1, Non-Tumor=0)

2.4. K-Nearest Neighbor Classifier

In this model, Euclidean distances are calculated between training and test samples, and distances are shorted. The first five closest data points are taken into consideration and how many points among these five belongs to which class. According to that, the test sample will assign to the class which has the most points in these five closest points. It is important to consider taking the odd number as the k value because in order to classification one of the class should dominate the other class in terms of data points in this k nearest neighbor. If k is equal to an even number, then there might be an equal amount of data points for each class in the nearest neighbor of the test sample, and classification wouldn't be successful.

```
[51]: # K-Nearest Neighbor Classification
      knn = KNeighborsClassifier(n_neighbors = 5,metric='minkowski',p=2) # KNN model for k=5 Nearest Neighbor and Eucledian Distance p=2
      knn.fit(X_train, Y_train)

[51]: KNeighborsClassifier()

[113]: #Checking Performance
       print('Accuracy of K-NN Classifier on Training Set:','%.2f'%(knn.score(X_train, Y_train)))
       print('Accuracy of K-NN Classifier on Test Set:','%.2f'%(knn.score(X_test, Y_test)))

       Accuracy of K-NN Classifier on Training Set: 0.97
       Accuracy of K-NN Classifier on Test Set: 0.95
```

Fig. 8. Application of K-Nearest Neighbor

For the K-Nearest Neighbor model, there is no optimal k value. It is a hyperparameter, and the best k can be found by testing different numbers. In this project 3,5,7,9,11,13,15,17,21,25,27 and 29 are tested to find best k number. Among all these, k=5 was one of the numbers that gave the best accuracy for the training and test datasets.

Fig. 9. shows that our model made 28 wrong predictions out of 319 tumors labeled data and made 8 wrong predictions out of 434 non-tumor labeled data.

```
[54]: Y_Predict_knn=knn.predict(X_test)

[55]: cm=np.array(confusion_matrix(Y_test,Y_Predict_knn,labels=[1,0]))
      confusion=pd.DataFrame(cm,index=['Tumor','Non-Tumor'],columns=['Predicted Tumor','Predicted Non-Tumor'])
      confusion
```

[55]:

	Predicted Tumor	Predicted Non-Tumor
Tumor	291	28
Non-Tumor	8	426

Fig. 9. K-Nearest Neighbor Prediction Result

Fig. 10. shows the decision region of K-Nearest Neighbor, the blue area is Non-Tumor class and, orange area is Tumor class.

```
[58]: knn_report=classification_report(Y_test,Y_Predict_knn)

[59]: x= df[['PCA-1','PCA-2']].values
      y = df['Class'].astype(int).values

      plot_decision_regions(x, y, clf=knn, legend=2)
      plt.xlabel('PCA-1')
      plt.ylabel('PCA-2')
      plt.title('K-NN with k = 5')
      plt.show()
```

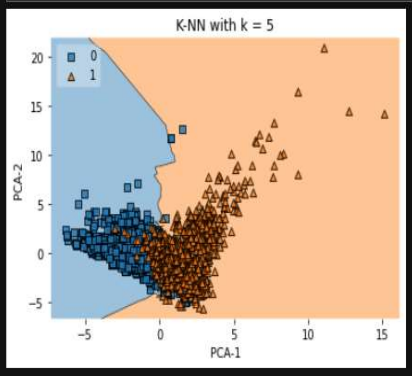


Fig. 10. Decision Boundary of K-Nearest Neighbor Model

2.5. Support Vector Machine Classifier

This algorithm aims to find hyperplanes that best separate the two classes. Distance between the hyperplane and closest data point is called margin, and these closest data points are called support vectors. For finding the best decision boundary, the distance should be maximum between the hyperplane and the support vectors. In this project, linear, polynomial, sigmoid, and radial basis functions (RBF) were tested, and among all these kernel functions, RBF performed the best decision boundary, and it is displayed in Fig. 13.

```
[85]: # Support Vector Machine Classification
      svm=SVC(kernel='rbf')

[86]: svm.fit(X_train,Y_train)

[86]: SVC()

[112]: #Checking Performance
       print('Accuracy of SVM Classifier on Training Set:','%.2f'%(svm.score(X_train, Y_train)))
       print('Accuracy of SVM Classifier on Test Set:','%.2f'%(svm.score(X_test, Y_test)))

       Accuracy of SVM Classifier on Training Set: 0.96
       Accuracy of SVM Classifier on Test Set: 0.96
```

Fig. 11. Application of Support Vector Machine

Fig. 12. shows that the SVM model made 28 wrong predictions out of 319 tumor labeled data and made 3 wrong predictions out of 434 non-tumor labeled data. Comparing Predicted Tumor columns with Fig. 9., SVM is made 5 more correct predictions than KNN.

```
[121]: Y_Predict_svm=svm.predict(X_test)

[122]: cm=np.array(confusion_matrix(Y_test,Y_Predict_svm,labels=[1,0]))
       confusion=pd.DataFrame(cm,index=['Tumor','Non-Tumor'],columns=['Predicted Tumor','Predicted Non-Tumor'])
       confusion
```

[122]:

	Predicted Tumor	Predicted Non-Tumor
Tumor	291	28
Non-Tumor	3	431

Fig. 12. Support Vector Machine Prediction Result

After seeing Fig. 10. and Fig. 13., Support Vector Machine gives a better decision boundary than K-Nearest Neighbor.

```
[123]: svm_report=classification_report(Y_test,Y_Predict_svm)

[124]: x= df[['PCA-1','PCA-2']].values
       y = df['Class'].astype(int).values

       plot_decision_regions(x, y, clf=svm, legend=2)
       plt.xlabel('PCA-1')
       plt.ylabel('PCA-2')
       plt.title('SVM Classification')
       plt.show()
```

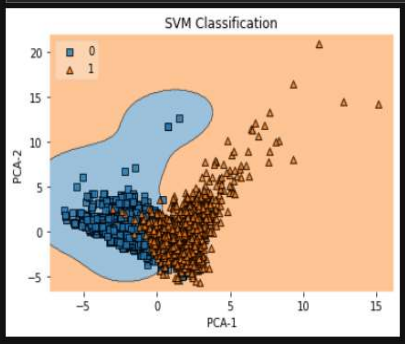


Fig. 13. Decision Boundary of Support Vector Machine

2.6. Naïve Bayes Classifier

Naïve Bayes algorithm is probabilistic approach for classification. It assumes that features PCA-1 and PCA-2 make an independent contribution to the probability of whether the image has a Tumor. In this project, a normal distribution assumption is made before starting the classification. For this reason, the Gaussian Naïve Bayes algorithm is used for classification.

```
[150]: # Naive Bayes Classification
       naive_bayes=GaussianNB()

[151]: naive_bayes.fit(X_train,Y_train)

[151]: GaussianNB()

[158]: #Checking Performance
       print('Accuracy of Naive Bayes Classifier on Training Set:','%.2f'%(naive_bayes.score(X_train, Y_train)))
       print('Accuracy of Naive Bayes Classifier on Test Set:','%.2f'%(naive_bayes.score(X_test, Y_test)))

       Accuracy of Naive Bayes Classifier on Training Set: 0.95
       Accuracy of Naive Bayes Classifier on Test Set: 0.96
```

Fig. 14. Application of Naïve Bayes Classification

Fig. 12. shows that the Naïve Bayes algorithm performs better in terms of classifying Non-Tumor labeled data. KNN and SVM algorithms made 28 wrong predictions from the test dataset. On the other hand, the Naïve Bayes algorithm made 18 wrong predictions.

```
[178]: Y_Predict_naive_bayes=naive_bayes.predict(X_test)

[179]: cm=np.array(confusion_matrix(Y_test,Y_Predict_naive_bayes,labels=[1,0]))
       confusion=pd.DataFrame(cm,index=['Tumor','Non-Tumor'],columns=['Predicted Tumor','Predicted Non-Tumor'])
       confusion
```

[179]:

	Predicted Tumor	Predicted Non-Tumor
Tumor	337	18
Non-Tumor	14	384

Fig. 15. Naïve Bayes Classification Prediction Result

```
[180]: naive_bayes_report=classification_report(Y_test,Y_Predict_naive_bayes)

[181]: x= df[['PCA-1','PCA-2']].values
       y = df['Class'].astype(int).values

       plot_decision_regions(x, y, clf=naive_bayes, legend=2)
       plt.xlabel('PCA-1')
       plt.ylabel('PCA-2')
       plt.title('Naive Bayes Classification')
       plt.show()
```

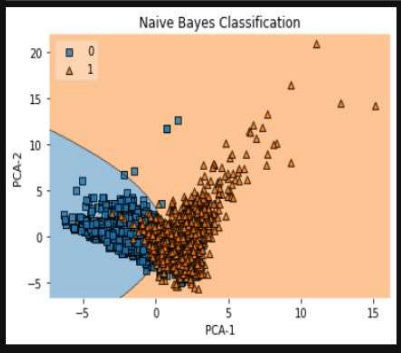


Fig. 16. Decision Boundary of Naïve Bayes Algorithm

According to Fig. 10., Fig. 13. And Fig. 16., the SVM classification model gives a better decision boundary than KNN and Naïve Bayes Algorithm. There are some points in Fig. 10. and Fig. 16. that Tumor labeled blue points are misclassified.

3. Results and Outcomes

Classification report shows three main metrics, which are precision, recall, and f1-score of the classification model. These metrics are calculated by True Positive, False Positive, True Negative, and False Negative.

True Positive (TP): If class label were 1 and predicted 1.

True Negative (TN): If class label were 0 and predicted 0.

False Positive (FP): If class label were 0 and predicted 1.

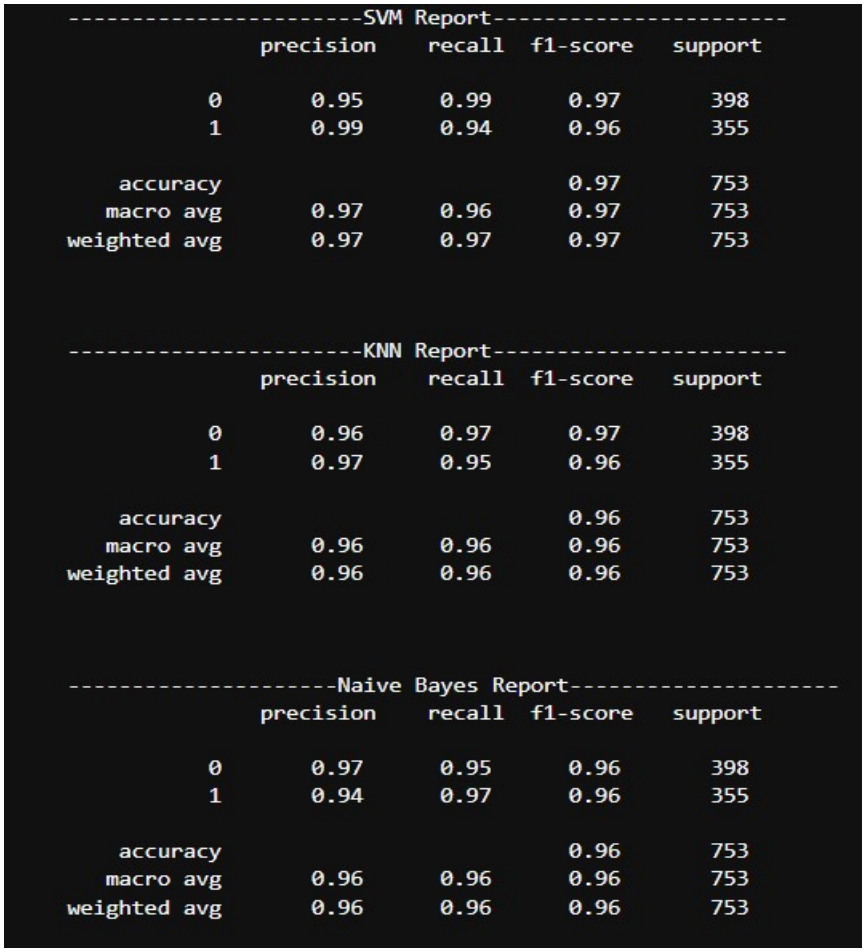
False Negative (FN): If class label were 1 and predicted 0.

Precision: Accuracy of positive predictions (Precision = TP/ (TP + FP)).

Recall: Fraction of positives that were correctly identified (Recall=TP/(TP+FN)).

F1-Score: Percentage of the prediction correctness (F1 Score = 2*(Recall * Precision) / (Recall + Precision)).

Support: “Number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores.” (Kohli, 2019)



----------SVM Report----------

	precision	recall	f1-score	support
0	0.95	0.99	0.97	398
1	0.99	0.94	0.96	355
accuracy			0.97	753
macro avg	0.97	0.96	0.97	753
weighted avg	0.97	0.97	0.97	753

----------KNN Report----------

	precision	recall	f1-score	support
0	0.96	0.97	0.97	398
1	0.97	0.95	0.96	355
accuracy			0.96	753
macro avg	0.96	0.96	0.96	753
weighted avg	0.96	0.96	0.96	753

----------Naive Bayes Report----------

	precision	recall	f1-score	support
0	0.97	0.95	0.96	398
1	0.94	0.97	0.96	355
accuracy			0.96	753
macro avg	0.96	0.96	0.96	753
weighted avg	0.96	0.96	0.96	753

Fig. 17. Classification Report

Fig. 17. shows that SVM performs better than other classification methods in terms of classification of the Tumor detection (Precision=0.99 for Tumor=1, precision value is bigger than other algorithms), it is already shown in Fig. 13. that it has a better decision boundary than other algorithms. On the other hand, Naïve Bayes performs better in terms of detecting Non-Tumor data (Precision=0.97 for Non-Tumor=0, precision value is bigger than other algorithms) and Fig. 15. supports this.

Generally, all three classifiers show high accuracy for brain tumor detection. However, based on f1-scores that SVM and KNN algorithms show equal performance and are more accurate than Naïve Bayes for Brain Tumor Detection. These methods can help doctors , neurosurgeons and radiologists to diagnose brain tumor from MRI image features.

4.Discussion

Our result shows that the Naïve Bayes Classifier detection rate in non-tumor images is 97% and 94% detection rate on tumor images. Our results are supported by the study of Zaw, H. T., Maneerat, N., & Win, K. Y. (2019), where they also showed Naïve Bayes Method for classification has an 81.25% detection rate on tumor images and a 100% detection rate in non-tumor images. (Zaw, Maneerat, Win,2019)

Comparing the SVM classification report with KNN, we can say that the SVM (99%) algorithm is more precise than the KNN (97%) algorithm in terms of detecting tumors on MRI images. On the other hand, the KNN (96%) algorithm is more precise than the SVM (95%) algorithm in terms of classifying non-tumor images. However, there are not many differences between them. These results were also supported by the study of Ramteke, R. J., & Monali, K. Y. (2012), where they also showed that for precision for detecting tumor, their KNN classifier has an 80% precision rate and the SVM classifier with RBF kernel has 80% precision for detecting the tumor on MRI images. (Ramteke, Monali,2012)

4.Running the Code

Install all the necessary libraries for python (Uncomment the code by deleting # after that run the code it will automatically install the necessary packages for you).

```
[ ]: # Installation of Necessary Pacages
     #!pip install numpy
     #!pip install -U matplotlib
     #!pip install seaborn
     #!pip install -U scikit-learn
     #!pip install pandas
     #!pip install opencv-python
     #!pip install os-sys
     #!pip install pip install mlxtend
```

Download the dataset from: https://www.kaggle.com/jakeshbohaju/brain-tumor/download

Copy the source code inside the dataset file which has “Brain Tumor.csv”. After that click the run button.

5.Referrances

Bohaju, J. (2020). Brain Tumor.

J. (2020). Train-Test Split for Evaluating Machine Learning Algorithms.

Kohli, S. (2019). Understanding a classification report for your machine learning model. India: Medium. com.

Nguyen, P. T., Shankar, K., Hashim, W., & Maseleno, A. (2019). Brain tumor segmentation and classification using KNN algorithm.

Ramteke, R. J., & Monali, K. Y. (2012). Automatic medical image classification and abnormality detection using k-nearest neighbour. International Journal of Advanced Computer Research, 2(4), 190.

Zaw, H. T., Maneerat, N., & Win, K. Y. (2019, July). Brain tumor detection based on Naïve Bayes Classification. In 2019 5th International Conference on Engineering, Applied Sciences and Technology (ICEAST) (pp. 1-4). IEEE.