INFO 7390 Research Project

Vignesh Murali

Section 06

NUID: 001886775

Abstract

Alzheimer's disease is a degenerative brain disorder which impairs a person's cognitive abilities and memory power. Early detection of this illness can help in better treatment and further research about the disease. Machine learning can be used to classify patients into groups of people who suffer from the illness and those who don't, depending on neurological imaging data and other socioeconomic factors. The aim of the research conducted in this project is to identify the best possible classification model to detect the presence of Alzheimer's Disease in new patients. The secondary aim of this project is to find out if traditional Machine Learning classifiers outperform neural network classifiers and deep learning networks. After training six conventional classifiers such as Support Vector Machines and Random Forests and neural network classifiers such as a Multilayer Perceptron and a deep learning model using Keras Classifier, the results obtained strongly suggest that traditional classifiers perform better than the artificial neural network models with the Random Forest Classifier achieving the highest accuracy with 92.55% while the deep learning network attained 91.48% accuracy. The primary reason for these results could possibly be the small size of the dataset which leads to overfitting of the neural network models. The findings related to this project could provide vital information for the detection of dementia using machine learning in the future and medical research.

Introduction

Alzheimer's is the most common form of dementia plaguing the world today. It is a progressive brain disorder which severely impairs a person's memory and cognitive abilities. The most horrifying aspect of this disorder is the fact that it is irreversible, and treatment can only provide limited relief. With over 5.7 million Americans living with dementia in 2018 and over 200,000 of them being under the age of 65, it is important to detect the onset of this disease at an early phase, so it can be treated and controlled before it is too late.

Machine learning has been widely used in the field of medicine only recently, by analyzing vital signs and other bodily characteristics we may be able to predict the onset of certain diseases such as cancer. There is massive potential in using these principles for identifying brain-related illnesses such as Alzheimer's

Disease because the volatility of the occurrence of these diseases is far lesser than most other diseases because most of the factors which contribute to these diseases can be identified numerically.

Various neurological factors such as the intercranial volume and brain volume, scores on psychological examinations such as the mini-mental state examination and clinical dementia rating as well as socioeconomic factors such as education level and age can all be used to prognose the onset of Alzheimer's disease. While reviewing the work of other scientists who attempted to classify Alzheimer's, it is easy to point out that most of them used only neurological data for their experiments and failed to consider the socioeconomic factors, which I have used in my analysis to verify their significance.

The primary aim of this project is to identify the most efficient machine learning model to predict the presence of dementia in patients and fine tune the model to obtain the highest accuracy. For a problem of this nature, where we are attempting to construct models which use training data to predict a labeled outcome based on existing patient records and neuro-imaging data, this is called supervised learning.

Methods

Dataset

The publicly available dataset used for this project has been obtained from OASIS (Open Access Series of Imaging Studies) - https://www.oasis-brains.org/. This is an organization aimed at providing neuroimaging datasets to the scientific community for further research. The dataset contains 150 subjects aged between 60 and 96 over one year for 373 image sessions, they are then classified into groups, namely – Demented, nondemented and converted. These groups are the classes which are to be predicted.

There are various conventional supervised learning classifiers which have been used in this project to classify patients as 'Demented', 'Nondemented or 'Converted'.

	Subject ID	MRI ID	Group	Visit	MR Delay	M/F	Hand	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
0	OAS2_0001	OAS2_0001_MR1	Nondemented	1	0	M	R	87	14	2.0	27.0	0.0	1987	0.696	0.883
1	OAS2_0001	OAS2_0001_MR2	Nondemented	2	457	M	R	88	14	2.0	30.0	0.0	2004	0.681	0.876
2	OAS2_0002	OAS2_0002_MR1	Demented	1	0	M	R	75	12	NaN	23.0	0.5	1678	0.736	1.046
3	OAS2_0002	OAS2_0002_MR2	Demented	2	560	M	R	76	12	NaN	28.0	0.5	1738	0.713	1.010
4	OAS2_0002	OAS2_0002_MR3	Demented	3	1895	M	R	80	12	NaN	22.0	0.5	1698	0.701	1.034

Feature Selection

Before these models are applied, the first order of business is to select the best possible features which can be used to for classification.

We may accomplish this by ranking features based on their importance with a random forest of decision trees as shown below.

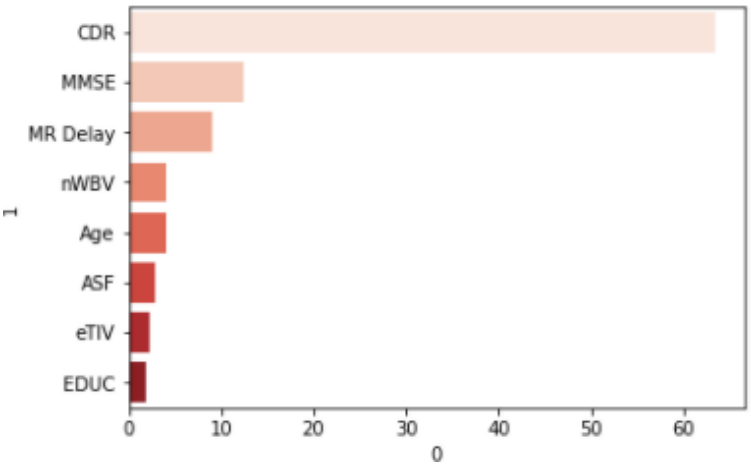
```
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier(n_estimators=40, max_depth=5, random_state=1,max_features=5)
```

```
random_forest.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features=5, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=40, n_jobs=1,
            oob_score=False, random_state=1, verbose=0, warm_start=False)
```

```
importances=100*random_forest.feature_importances_
sorted_feature_importance = sorted(zip(importances, list(X_train)), reverse=True)
features_pd = pd.DataFrame(sorted_feature_importance)
print(features_pd)
```

```
           0         1
0  63.501291       CDR
1  12.377521      MMSE
2   8.972169  MR Delay
3   4.064768      nWBV
4   4.039277       Age
5   2.810986       ASF
6   2.342095      eTIV
7   1.891893      EDUC
```



We may eliminate the 3 lowest features as the rest are not as significantly low in importance as them.

The next step is to train our classification models with our data.

Code with documentation

Supervised learning is defined as the task of learning a function that maps an input to an output based on a set of input-output pairs. In this project, several supervised learning models have been implemented, namely:

Conventional Machine Learning Classifiers:

- **Random Forests Classifier**: A random forest is a meta estimator that fits several decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. When 5 features are considered for the best split, we obtain the best results for our dataset.

```
In [90]: from sklearn.ensemble import RandomForestClassifier

         random_forest = RandomForestClassifier(n_estimators=40, max_depth=5, random_state=1,max_features=5)

In [91]:
         random_forest.fit(X_train, y_train)

Out[91]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=5, max_features=5, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=40, n_jobs=1,
                     oob_score=False, random_state=1, verbose=0, warm_start=False)
```

- **Decision Tree Classifier**: Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). Max features is selected as 5, accuracy plateaus after this value.

```
In [100]: from sklearn.tree import DecisionTreeClassifier
          dectree = DecisionTreeClassifier(max_features=5)

In [101]: dectree.fit(X_train, y_train)

Out[101]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                      max_features=5, max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

- **K Nearest Neighbors Classifier**: An object is assigned to a class with K of its nearest neighbors using a similarity measure or distance function. Accuracy remains the same with all three available distance functions – Manhattan, Euclidean and Minkowski. 8 neighbors selected because the network plateaus when it exceeds this value.

```
In [96]: from sklearn.neighbors import KNeighborsClassifier
         nneighbor = KNeighborsClassifier(n_neighbors=8,metric='euclidean')

In [97]: nneighbor.fit(X_train2, y_train2)

Out[97]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='euclidean',
                     metric_params=None, n_jobs=1, n_neighbors=8, p=2,
                     weights='uniform')
```

- **Naïve Baye's Classifier**: Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class. The only parameter is priors which means prior probabilities for each class. It is advised to leave this parameter as none because priors are automatically calculated by the algorithm based on the data.

```
In [104]: from sklearn.naive_bayes import GaussianNB
          gnb = GaussianNB()

In [105]: gnb.fit(X_train,y_train)

Out[105]: GaussianNB(priors=None)
```

- **Support Vector Classifier**: A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. C is the penalty parameter and it is set to 2 because the accuracy somehow increases, and the kernel is set to linear, the other kernels (poly and rbf) are either too slow or inaccurate.

```
In [108]: from sklearn.svm import SVC
          supvc = SVC(kernel='linear',C=2)

In [109]: supvc.fit(X_train2,y_train2)

Out[109]: SVC(C=2, cache_size=200, class_weight=None, coef0=0.0,
            decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
            max_iter=-1, probability=False, random_state=None, shrinking=True,
            tol=0.001, verbose=False)
```

- **Ada Boost Classifier**: Ada-boost classifier combines weak classifier algorithm to form strong classifier. A single algorithm may classify the objects poorly. But if we combine multiple classifiers with selection of training set at every iteration and assigning right amount of weight in final voting, we can have good accuracy score for overall classifier. SAMME is a boosting algorithm which works better for multiclass classification, SAMME.R works is conventionally used for binary classification problems.

```
In [112]: from sklearn.ensemble import AdaBoostClassifier
          abc = AdaBoostClassifier(algorithm='SAMME')

In [113]: abc.fit(X_train2,y_train2)

Out[113]: AdaBoostClassifier(algorithm='SAMME', base_estimator=None, learning_rate=1.0,
                    n_estimators=50, random_state=None)
```

Neural Network Classifiers:

- **Multilayered Perceptron Classifier**: Multilayer perceptron classifier (MLPC) is a classifier based on the feedforward artificial neural network. MLPC consists of multiple layers of nodes. Each layer is fully connected to the next layer in the network. Nodes in the input layer represent the input data. All other nodes map inputs to outputs by a linear combination of the inputs with the node's weights w and bias b and applying an activation function.
  According to Scikit-learn's documentation, the solver (used for weight optimization) 'lbfgs' is more appropriate for smaller datasets compared to other solvers such as 'adam'. Three layers of nodes are used in this model.

```
In [75]: from sklearn.neural_network import MLPClassifier

In [77]: mlp = MLPClassifier(max_iter=500,solver='lbfgs',hidden_layer_sizes=(10,30,20),activation='tanh')

In [78]: mlp.fit(X_train2,y_train2)

Out[78]: MLPClassifier(activation='tanh', alpha=0.0001, batch_size='auto', beta_1=0.9,
                       beta_2=0.999, early_stopping=False, epsilon=1e-08,
                       hidden_layer_sizes=(10, 30, 20), learning_rate='constant',
                       learning_rate_init=0.001, max_iter=500, momentum=0.9,
                       nesterovs_momentum=True, power_t=0.5, random_state=None,
                       shuffle=True, solver='lbfgs', tol=0.0001, validation_fraction=0.1,
                       verbose=False, warm_start=False)
```

- **Deep Neural Network Using Keras Classifier**: The goal of a feedforward artificial neural network is to approximate some function f*. For example, for a classifier, y = f*(x) maps an input x to a class labeled as y. These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y.
  The Keras library provides a convenient wrapper for deep learning models to be used as classification or regression estimator s in scikit-learn. The KerasClassifier class in Keras take an argument "build_fn" which is the name of the function to call to get your model. You must define a function that defines your model, compiles it and returns it.

```
In [100]: from sklearn.preprocessing import LabelEncoder
          from sklearn.preprocessing import OneHotEncoder, LabelBinarizer

          lb = LabelBinarizer()
          y_train3 =lb.fit_transform(y_train2)

In [106]: def baseline_model():
              classifier = Sequential()

          # Adding the input layer and the first hidden layer
              classifier.add(Dense(activation = 'relu',  input_dim = 8, units = 8, kernel_initializer = 'uniform'))
          # Adding the second hidden layer
              classifier.add(Dense( activation = 'relu', units = 15, kernel_initializer = 'uniform'))
          # Adding the third hidden layer
              classifier.add(Dense( activation = 'relu', units = 15, kernel_initializer = 'uniform'))

          # Adding the output layer
              classifier.add(Dense(activation = 'sigmoid', units = 3, kernel_initializer = 'uniform' ))

          # Compiling the ANN
              classifier.compile(optimizer = 'adamax', loss = 'categorical_crossentropy', metrics = ['accuracy'])


          # Fitting the ANN to the Training set
              return classifier

In [107]: estimator = KerasClassifier(build_fn=baseline_model, epochs=150, batch_size=5, verbose=0)

In [108]: estimator.fit(X_train2, y_train2)

Out[108]: <keras.callbacks.History at 0x1f0348dee80>
```

There are three hidden layers in this network, with 8,15 and 15 nodes respectively. The output layer is activated by a sigmoid function because it tends to bring the activations to either side of the curve (above x = 2 and below x = -2 for example) which helps in making clear distinctions for classification of data.

Validating the accuracy of our models: For viewing the accuracy of our classification models, we use the Sci-kit learn metrics function "Accuracy_Score", which checks the accuracy of our predicted class labels for our test feature set against the test labels.

A confusion matrix is then applied to view the actual number of predicted and true classes. It has been implemented as follows.

```
In [38]: y_predict = gnb.predict(X_test)
         nbscore = 100*accuracy_score(y_test, y_predict)
         print('Accuracy of Naive Bayes Classifier is ')
         print(100*accuracy_score(y_test,y_predict))

         Accuracy of Naive Bayes Classifier is
         90.42553191489363
```

```
In [39]: pd.DataFrame(
             confusion_matrix(y_test, y_predict),
             columns=['Predicted Healthy', 'Predicted alzheimers','Predicted Converted'],
             index=['True Healthy', 'True alzheimers','True converted']
         )
```

Out[39]:

	Predicted Healthy	Predicted alzheimers	Predicted Converted
True Healthy	2	1	6
True alzheimers	2	33	0
True converted	0	0	50

Just as the accuracy is calculated for the Naïve Baye's classifier in the above screenshot, it is calculated for all the other classification models used in this project in the same manner.

Results

The accuracy scores obtained from all the classification models used and the respective confusion matrices are attached in this section.

<table>
<tr><th>Classification Model</th><th>Accuracy</th></tr>
<tr><td>Random Forest Classifier
<table><tr><th></th><th>Predicted Healthy</th><th>Predicted Alzheimers</th><th>Predicted Converted</th></tr><tr><td>True Healthy</td><td>3</td><td>1</td><td>5</td></tr><tr><td>True Alzheimers</td><td>1</td><td>34</td><td>0</td></tr><tr><td>True converted</td><td>0</td><td>0</td><td>50</td></tr></table></td><td>92.55%</td></tr>
<tr><td>Support Vector Classifier
<table><tr><th></th><th>Predicted Healthy</th><th>Predicted Alzheimers</th><th>Predicted Converted</th></tr><tr><td>True Healthy</td><td>2</td><td>1</td><td>6</td></tr><tr><td>True Alzheimers</td><td>0</td><td>35</td><td>0</td></tr><tr><td>True converted</td><td>0</td><td>0</td><td>50</td></tr></table></td><td>92.55%</td></tr>
<tr><td>Keras Classifier using a Deep Neural Network**
<table><tr><th></th><th>Predicted Healthy</th><th>Predicted Alzheimers</th><th>Predicted Converted</th></tr><tr><td>True Healthy</td><td>2</td><td>1</td><td>6</td></tr><tr><td>True alzheimers</td><td>1</td><td>34</td><td>0</td></tr><tr><td>True converted</td><td>0</td><td>0</td><td>50</td></tr></table></td><td>91.48%</td></tr>
<tr><td>Ada Boost Classifier</td><td>90.42%</td></tr>
</table>

<table><tr><th></th><th>Predicted Healthy</th><th>Predicted Alzheimers</th><th>Predicted Converted</th></tr><tr><td>True Healthy</td><td>0</td><td>3</td><td>6</td></tr><tr><td>True alzheimers</td><td>0</td><td>35</td><td>0</td></tr><tr><td>True converted</td><td>0</td><td>0</td><td>50</td></tr></table>	
Naïve Baye's Classifier <table><tr><th></th><th>Predicted Healthy</th><th>Predicted alzheimers</th><th>Predicted Converted</th></tr><tr><td>True Healthy</td><td>2</td><td>1</td><td>6</td></tr><tr><td>True alzheimers</td><td>2</td><td>33</td><td>0</td></tr><tr><td>True converted</td><td>0</td><td>0</td><td>50</td></tr></table>	90.42%
K Nearest Neighbors Classifier <table><tr><th></th><th>Predicted Healthy</th><th>Predicted Alzheimers</th><th>Predicted Converted</th></tr><tr><td>True Healthy</td><td>0</td><td>3</td><td>6</td></tr><tr><td>True Alzheimers</td><td>1</td><td>34</td><td>0</td></tr><tr><td>True converted</td><td>1</td><td>0</td><td>49</td></tr></table>	88.29%
Multi-Layered Perceptron** <table><tr><th></th><th>Predicted Healthy</th><th>Predicted Alzheimers</th><th>Predicted Converted</th></tr><tr><td>True Healthy</td><td>2</td><td>1</td><td>6</td></tr><tr><td>True alzheimers</td><td>1</td><td>34</td><td>0</td></tr><tr><td>True converted</td><td>8</td><td>1</td><td>41</td></tr></table>	81.91%
Decision Tree Classifier <table><tr><th></th><th>Predicted Healthy</th><th>Predicted Alzheimers</th><th>Predicted Converted</th></tr><tr><td>True Healthy</td><td>3</td><td>1</td><td>5</td></tr><tr><td>True Alzheimers</td><td>6</td><td>29</td><td>0</td></tr><tr><td>True converted</td><td>9</td><td>0</td><td>41</td></tr></table>	77.65%

**Note: Neural networks are stochastic algorithms and may produce a different result each run.

Discussion

From our experiments, we can conclude that traditional machine learning algorithms such as Random Forests and Support Vector Machines aid in more efficient classification of data compared to deep learning neural networks. There are several possibilities for this outcome which will be discussed in this section.

Generally, artificial neural networks require huge amounts of data to get trained efficiently. Our dataset is relatively small compared to the data needed to train the network properly without the presence of overfitting.

Future Work:

- Perform optimized regularization or data augmentation to reduce overfitting on the neural network models.
- Further tune the parameters of the classifiers to improve accuracy.
- Run the networks on larger datasets.
- Identify classification models to prognose the presence of other brain-related illnesses such as Parkinson's disease and epileptic seizures.

References

- https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/
- https://www.kaggle.com/jboysen/mri-and-alzheimers
- Zhang, Yudong, et al. "Detection of subjects and brain regions related to Alzheimer's disease using 3D MRI scans based on eigenbrain and machine learning." *Frontiers in Computational Neuroscience* 9 (2015): 66.
- https://keras.io/getting-started/sequential-model-guide/
- http://scikit-learn.org/stable/modules/multiclass.html
- Desikan, Rahul S., et al. "Automated MRI measures identify individuals with mild cognitive impairment and Alzheimer's disease." *Brain* 132.8 (2009): 2048-2057.
- Reid, Andrew T., and Alan C. Evans. "Structural networks in Alzheimer's disease." *European neuropsychopharmacology*23.1 (2013): 63-77.