Course: 634
Subject: Final Project
Topic: 3 Algorithm Comparison
Name: Sheethal Mathew
Email:ssm29@njit.edu

3 Algorithms: Random Forest, KNN, LSTM

Contents

Introduction

'Option 1: Supervised Data Mining (Classification)' of 3 different algorithm is what I chose for this project. The objective of the project is to apply different learning models on the same dataset and analyze the result. ROC curve after the training and testing will be used to determine the ttest result. First two algorithm used are Random Forest and KNN classification. For the deep learning option; LSTM – is usually time series dependent and my dataset has no time value so that eliminates the use of LSTM and Bidirectional LSTM. Out of the 3 deep learning algorithm options, what I am left with is GRU

Random Forest

Divide the data into train data and test data.

Find the Gini Impurity of each attribute column against the label of the dataset.

Pick the column with highest gini value.

If an attribute has n different categories, then the data has to be split 2^{n-1}-1.

Keep growing n trees

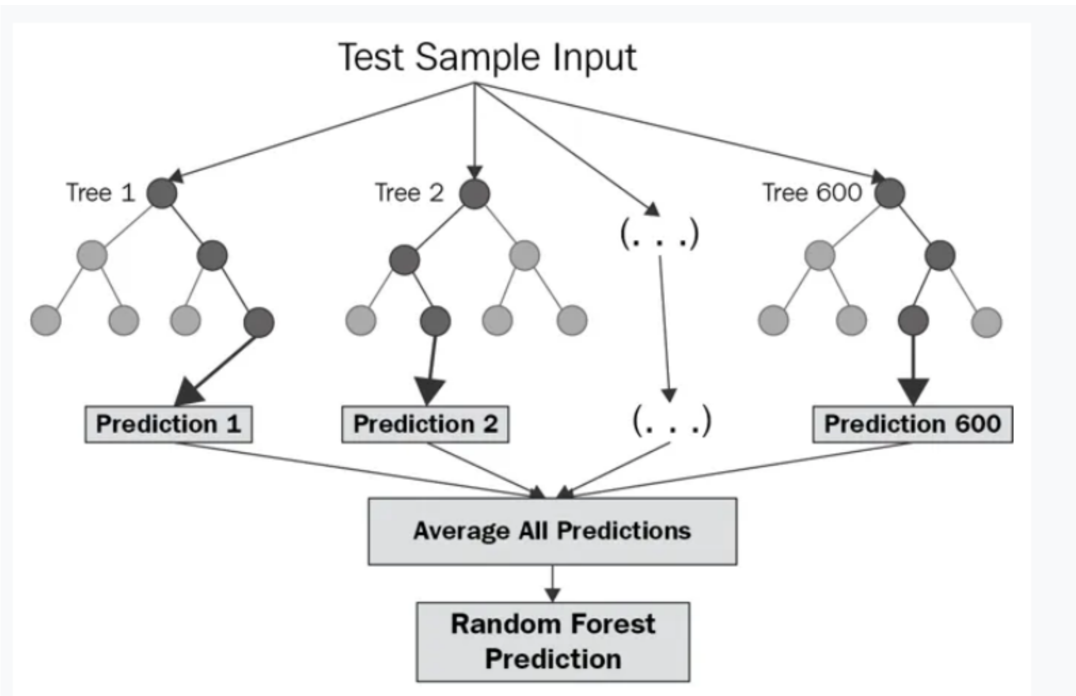

Figure 1: Random Forest Classification([Source1])

KNN (K-Nearest Neighbor)

The k value was picked by the datascientist.

Labels datapoints based on the rule of majority classification of nearest neighbors

K nearest neighbors are calculated using Euclidean distance

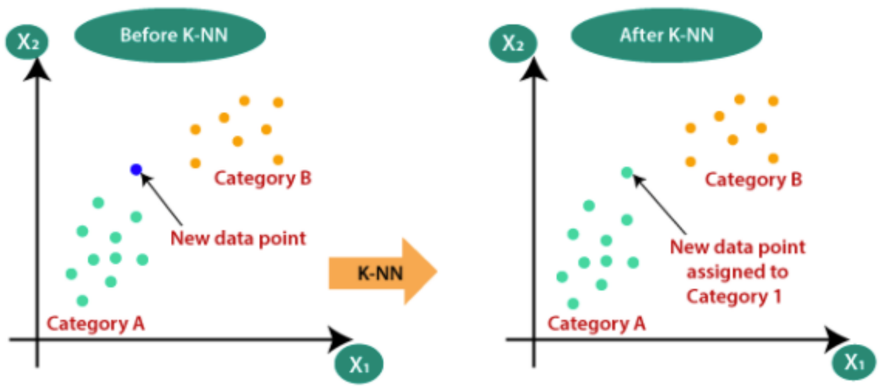

Figure 2: K-Nearest Neighbor Classification (Source2javapoint)

Long short term memory(LSTM)

LSTM is a recurrent neural network variant with long term memory. A lot of times LSTM is good with dataset that has a time component. This model has 3 gates, input, putput, and keep gate where data is maintained.

- the "Input" or "Write" Gate, which handles the writing of data into the information cell,
- the "Output" or "Read" Gate, which handles the sending of data back onto the Recurrent Network, and
- the "Keep" or "Forget" Gate, which handles the maintaining and modification of the data stored in the information cell.

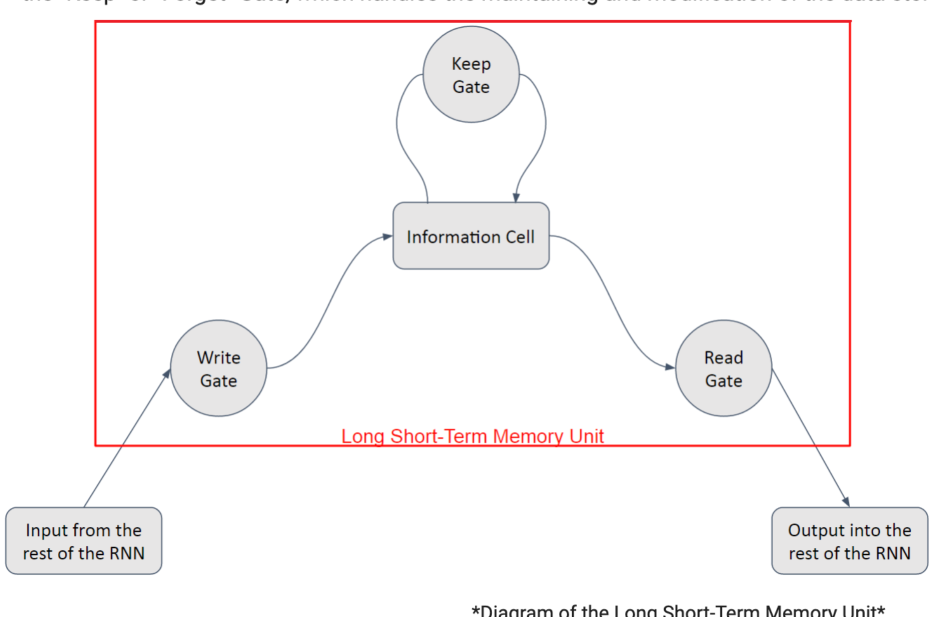

Figure 3: LSTM (source3lstm)

Assumption

Since my only options were LSTM and GRU and my dataset did not contain time specific dependent columns, I figured picking GRU was a better choice. It has 2 gates

Requirement

Software

Python3, https://colab.research.google.com/ was used to build and execute the code.
Pyenv environments are in built in the system.

Libraries used:
mlxtend.plotting
numpy
pandas
matplotlib.pyplot
sklearn.model_selection
sklearn.neighbors
RandomForestClassifier
KNeighborsClassifier
RandomForestRegressor
Export_graphviz
Metrics
Accuracy_score
confusion_matrix
plot_roc_curve
SVC
Seaborn
TensorFlow
Keras
MinMaxScaler
Sequential()

Hardware

MacBook Pro, Apple M1chip, MacOS 12.0.1

List of Source code Files

Mathew_Sheethal_634Option1.ipynb

Compile the source Code

Go to the google collab link here:
https://colab.research.google.com/drive/1vne05liWb6hsMCkkRltfapYEh348aSdq#scrollTo=d66_si_wE9pv
Hit connect.=
You will notice diabetes.csv preuploaded under files

How to run the Application
Click on RunTime on task bard -> Run All

Dataset

Dataset is taken from the "Pima Indians Diabetes Database"(Dataset)
The dataset is from National Institute of Diabetes and Digestive and Kidney diseases. Using the dataset, and analyzing it, it can predicted whether a patient has diabetes. It must be noted that the data is taken from Pima Indian Heritage women only. The attributes that result in does or does not have diabetes are Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, Age, DiabetesPedigreeFunction.
The label of the data is the 'Outcome' column which has values of 0 or 1 indicating no diabetes, and yes diabetes respectively.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Figure3: Diabetes dataset top 5 rows

For the deep learning model LSTM I realized my data is not good because there is no time series attribute associated with data.
To mitigate this I took the diabetes.csv and added timeseries column randomly to the data.
This will help me run the LSTM model.
Data CleanUp
The anomalies in the data be determined using the describe() method.
This method creates statistics summary of the tendency, dispersion and shape of a dataset distribution excluding NanValues. This method deals with numeric values.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Figure 4: Dataset Anomalies
It can be dound that Blood Pressure of 0, and Skin thickness of 0 and other 0.00 values are an anomaly. The only way to get a 0 for these metrics are if the person does not exist(null). To mitigate this the dataset has been altered to replace 0 with Nan values. Nan values are Not a Number special values in datafram or numpy arrays. So we mark 0 as data missing.
Skin Thickness and Insulin as 0.00 does not make sense. If skin thickness is 0 or insulin is 0 then it does not make sense that the person is alive.

The minimum is 0.00 for several of these attributes and that should be data that can be eliminated as it will skew our results down heavily.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 9 columns

Figure 5: Before NaN

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedi
0	6	148.0	72.0	35.0	125.0	33.6	
1	1	85.0	66.0	29.0	125.0	26.6	
2	8	183.0	64.0	29.0	125.0	23.3	
3	1	89.0	66.0	23.0	94.0	28.1	
4	0	137.0	40.0	35.0	168.0	43.1	
...	...	...	...	...	...	...	
763	10	101.0	76.0	48.0	180.0	32.9	
764	2	122.0	70.0	27.0	125.0	36.8	
765	5	121.0	72.0	23.0	112.0	26.2	
766	1	126.0	60.0	29.0	125.0	30.1	
767	1	93.0	70.0	31.0	125.0	30.4	

768 rows × 9 columns

Figure 6: After NaN

Split the data into test set and training set. The Need to split it before applying model is because if training set includes test set, that would mean the model would return 100% accuracy, but that defeats the purpose of analyzing and learning data.
For the Random Forest case and KNN, data was split to 25% test and 75% training set.
The two datafiles are: diabetes.csv and timeDiabetes.csv

Implementation and Result

Auc scores from RandomForest Model and KNN Model are 55.5% and 81.9% respectively. For Random Forest training it can be seen that the training ROC curve is more diagonal and less curvy than that of knn model with higher auc. Knn model has reached its almost stabilized form at 81.9% as represented by the graph. In my case, it was proven that KNN was more accurate.

The column Outcome determines the final results and therefore can be used for labelling the dataset as left pile and right pile.

Random Forest Result

After the dataset was split into trainingAttributes, testAttributes, Train Label and Test Label. randomModel was created using RandomForestClassifier with 100 number of trees. This n_estimator number was provided by me at random. The randomModel was applied on the sets: trainingAttributesSet, trainingLabelSet. Once the training is done, a prediction was done on test set using randModel.predict() function.

```
randModel = RandomForestClassifier(n_estimators=100,random_state=0) #n_estimators is number of trees in forest
randModel.fit(train_attributes, trainLabel)
predictedTest = randModel.predict(test_attributes)
```

Figure7: Applying RandomForestClassifier on dataset and training it, and predicting against Test set

The accuracy is 64%. Below you will find the tree with depth reduced to 3 levels instead of the 100 branch tree.

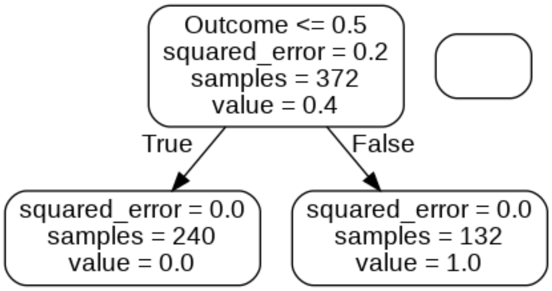

Figure 8: RandomForestTree with depth reduced to 3 levels.

Receiver Operating Characteristic Cruve(ROC) tells us how good model can differentiate between two classes. In our case, if the patient has diabetes or not. When the curve is lower it means it is able to determine the difference between diabetes and no diabetes. Figure below shows the ROC curve is slow and that means accuracy of prediction is high.

```
randModel = RandomForestClassifier(n_estimators=10, random_state=42)
randModel.fit(trainAttr, trainLabel)
ax = plt.gca()
randModel_disp = plot_roc_curve(randModel, testAttr, testLabel, ax=ax, alpha=0.8)
svc_disp.plot(ax=ax, alpha=0.8)
plt.show()
```

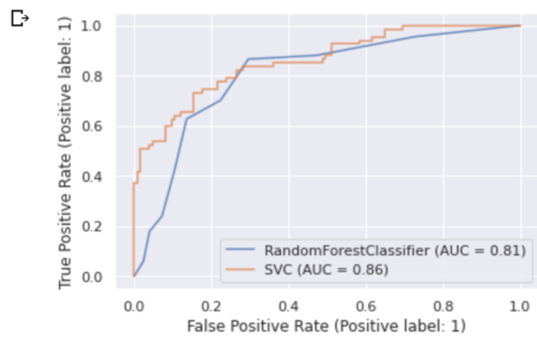

Figure 9: ROC curve of randomForestModel

KNN Result

Since Knn finds the Euclidean distance between neighboring datapoints, it is important to normalize the data to a range such that the Euclidean distance won't be skewed. In its original value, if some point is really far away(an anomaly) and weightange is not calculated correctly, that could skew the distance calculation significantly.

The normalizing of data to standard set is done using the following lines of code.

```
a = np.array(attributes)
scaler = preprocessing.StandardScaler()
#transform
standrd = scaler.fit_transform(a)
standrd
```

Now using the 'KNegihborsClassifier' library, knn model is applied to the trainingAttributes and trainingLabel that was created earlier when cleaning up the dataset.

After training on 75% of the data the max score is 77.64% and k=4,6

I atrbitarily chose k=7 for knn.
The following lines of code trained the dataset with knn model

```
for i in range(1,7):
  knn = KNeighborsClassifier(i)
  knn.fit(trainAttr, trainLabel)
```

```
[20] p = sns.lineplot(range(1,7), train_calc, marker='*', label='Train Calculated')
     p = sns.lineplot(range(1,7), test_calc, marker='o', label='Test Calculated')
```

Figure 10: KNN Visualized comparison of score on trained dataset vs test dataset
When the model is run on the labeled data, Euclidean distance is calculated between points, and point is classified into the same class as neighbors. It is important to normalize data so the Euclidean distance wont show skewed results. Hence over-fitting and underfitting problems can be well avoided with cross validation techniques

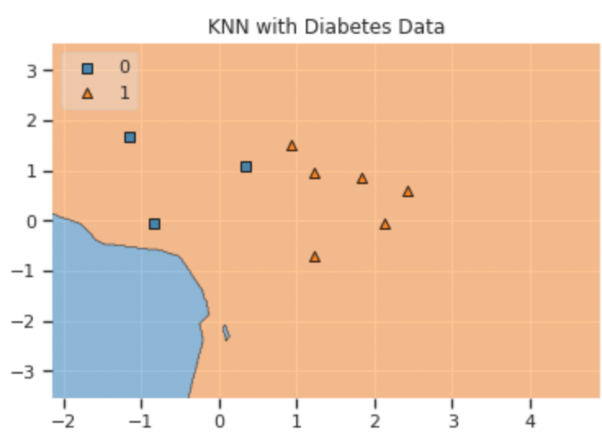

Figure 11: KNN model neighbors with k=7

Now we can run confusion matrix on the model. It takes into account predicted values vs actual values. The TN,FP,FN,TP can be found in the matrix below. Confusion matrices represent true labels on rows and predicted label on columns. Diagonal values represent the percent of predicted label matching true label.

In the graph below you will find support of TN is 142, FP is 25, FN is 35, and TP is 54.
The accuracy of the models is 77%

```
The KNN confusion matrix:
[[142  25]
 [ 35  54]]


The metrics KNN classification report:
              precision    recall  f1-score   support

           0       0.80      0.85      0.83       167
           1       0.68      0.61      0.64        89

    accuracy                           0.77       256
   macro avg       0.74      0.73      0.73       256
weighted avg       0.76      0.77      0.76       256
```

Figure 12: KNN confusion matrix and classification report

```
svc = SVC(random_state=42)
svc.fit(Attr_test, label_test)
svc_disp = plot_roc_curve(svc, Attr_test, label_test)
svc_disp.plot(ax=ax, alpha=0.8)
```

```
AUC score is:  0.8193500639171096
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f82e43d57d0>
```

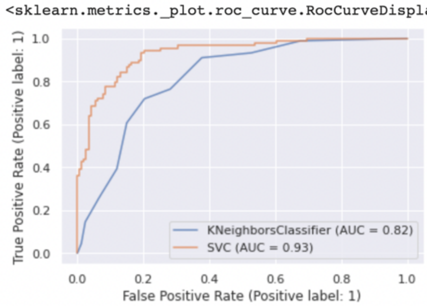

Figure 13: KNN ROC curve represents a bigger curve than the Random Forest model
Here there AUC value is 0.82 and the curve is higher for KNN model proving it is the better model than RandomForest.

LSTM model-Long Short Term Memory

Since data was maniupilated and time series column was added. The steps to finding anomaly and cleaning 0 with NaN values are repeated with the timeDiabetes.csv.
After splitting data into training and testing set, the data range is normalized using MinMaxScaler library. To work with LSTM model the data is then split to inputs and outputs.
As rerepsented in the code below, we first split to input and output then reshape the data.

```
train_X0, train_y0 = train[:split, 1:], train[:split, 0]
val_X0, val_y0 = train[split:,1:], train[split:,0]
test_X0, test_y0 = test[:, 1:], test[:, 0]

train_y=np.asarray(train_y0).reshape(( -1 , 1 ))
val_y=np.asarray(val_y0).reshape(( -1 , 1 ))
test_y=np.asarray(test_y0).reshape(( -1 , 1 ))
```

Finally after reshaping, the lstm data model is applied to sequential data and their mean squared error is calculated.

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_14 (LSTM)              (None, 5)                 140

 dense_8 (Dense)             (None, 1)                 6

=================================================================
Total params: 146
Trainable params: 146
Non-trainable params: 0
_________________________________________________________________
'lstm_model.add(LSTM(units=100, input_shape=(train_X.shape[1], train_X.shape[2]), return_sequences=True))\nlst
(LSTM(units=100, return_sequences=False))\nlstm_model.add(Dense(1))\nlstm_model.add(Activation("linear"))\n#ad
as.optimizers.adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)\nmodel.compile
se\', \n              optimizer = adam)\nmodel.summary()\ncallbacks = [\n        EarlyStopping(monitor=\'val_l
ence=8)\n        #ModelCheckpoint(filepath=save_fname, monitor=\'val_loss\', save_best_only=True)\n        ]'
```

Figure 14: LSTM model after input and output are well defined.

```
lstm_model.fit(X_train, y_train, epochs=10, batch_size=1)
```

The line above is lstm model can fit to the dataset.
It returned epoch of 10 layers and this is where the codestopped. I am unable to move past this point.
The error I see is "`ValueError: Creating variables on a non-first call to a function decorated with tf.function.". So I am unable to move past the error to compare the LSTM model with the others. I need to be able to plot the graph before computing the ROC curve.`

References

https://www.kaggle.com/uciml/pima-indians-diabetes-database?select=diabetes.csv

https://medium.com/analytics-vidhya/evaluating-a-random-forest-model-9d165595ad56

https://www.kaggle.com/shrutimechlearn/step-by-step-diabetes-classification-knn-detailed/notebook *https://scikit-learn.org/0.24/auto_examples/miscellaneous/plot_roc_curve_visualization_api.html*

https://colab.research.google.com/github/Gurubux/CognitiveClass-DL/blob/master/2_Deep_Learning_with_TensorFlow/DL_CC_2_3_RNN/3.4-Review-LSTM-MNIST-Database.ipynb