INFORAMATION SECURITY ANALYSIS AND AUDIT

Topic: DATA SECURE SMART HOME AUTOMATION SYSTEM

NAME-SATRUNJAY KUMAR

REG NO-18BIT0040

SLOT -G1

REVIEW 3 (COMPARATIVE ANALYSIS)

Link for the integrated part (all team members) video:

https://youtu.be/z0bK9h3Qz10

Link for my review 3 video(individual part):
https://drive.google.com/file/d/1w419fwZGItKh4J-VMER4am4alofdfZzU/view?usp=sharing

Link for my other necessary files:

https://github.com/satrunjaykumar/Nasscom-project.git

Comparative Analysis

Here, I am going to compare 3 different Models that I have created for the Classification of the text into commands.

1st model (CNN model without Word2Vec Layer)

```
#model without Word2vec
def new_model():
    taskbrain=models.Sequential()
    taskbrain.add(layers.Dense(32, activation="relu", input_shape=(444,)))
    taskbrain.add(layers.Dense(32))
    taskbrain.add(Dropout(0.2))
    taskbrain.add(layers.Dense(64))
    taskbrain.add(Dropout(0.4))
    taskbrain.add(layers.Dense(8,activation='softmax'))
    taskbrain.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy','mae','mse'])
    return taskbrain
```

2nd model (ANN model with Word2Vec Layer and Deep network)

```
def build_model():
    taskbrain=models.Sequential()
    taskbrain.add(word2vec)
    taskbrain.add(layers.Dense(16))
    taskbrain.add(Dropout(0.2))
    taskbrain.add(layers.Dense(32))
    taskbrain.add(Dropout(0.2))
    taskbrain.add(layers.Dense(64))
    taskbrain.add(Dropout(0.4))
    taskbrain.add(layers.Dense(128))
    taskbrain.add(Dropout(0.4))
    taskbrain.add(layers.Dense(64))
    taskbrain.add(Dropout(0.2))
    taskbrain.add(layers.Dense(8,activation='softmax',input_shape=(20,)))
    taskbrain.compile(optimizer='rmsprop',loss='categorical_crossentropy',metrics=['accuracy','mae','mse'])
    return taskbrain
```

3rd model (CNN model with Word2Vec and Shallow network)

```
def build_model1():
    taskbrain=models.Sequential()
    taskbrain.add(word2vec)
    taskbrain.add(layers.Dense(20))
    taskbrain.add(Dropout(0.3))
    taskbrain.add(layers.Dense(8,activation='softmax',input_shape=(20,)))
    taskbrain.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy','mae','mse'])
    return taskbrain
```

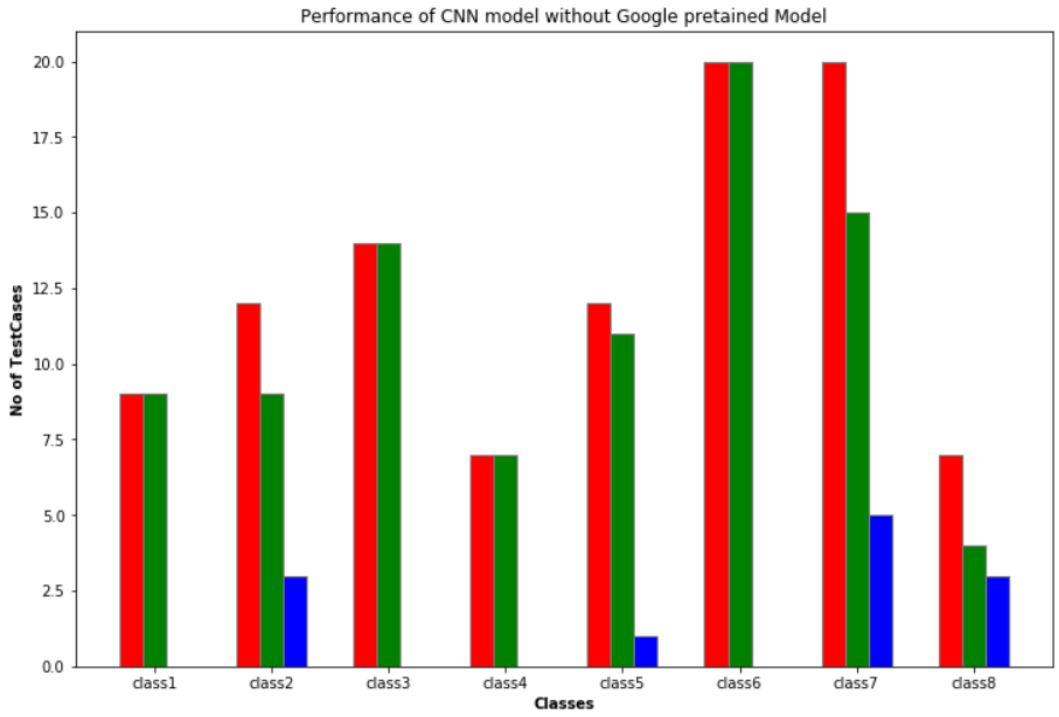
Tabular Comparison of All models

	F1_score	Precision_score	Accuracy_Score
#model1 (CNN without Word2Vec pretrained Model)	0.8823113957111316	0.8921400034740316	0.8811881188118812
#model2 (ANN with Word2Vec pretrained Model and deep network)	0.9215839209217587	0.9294529452945294	0.9207920792079208
#model3 (CNN with Word2Vec pretrained Model and Shallow network)	0.9515301283822964	0.9655565556555655	0.9504950495049505

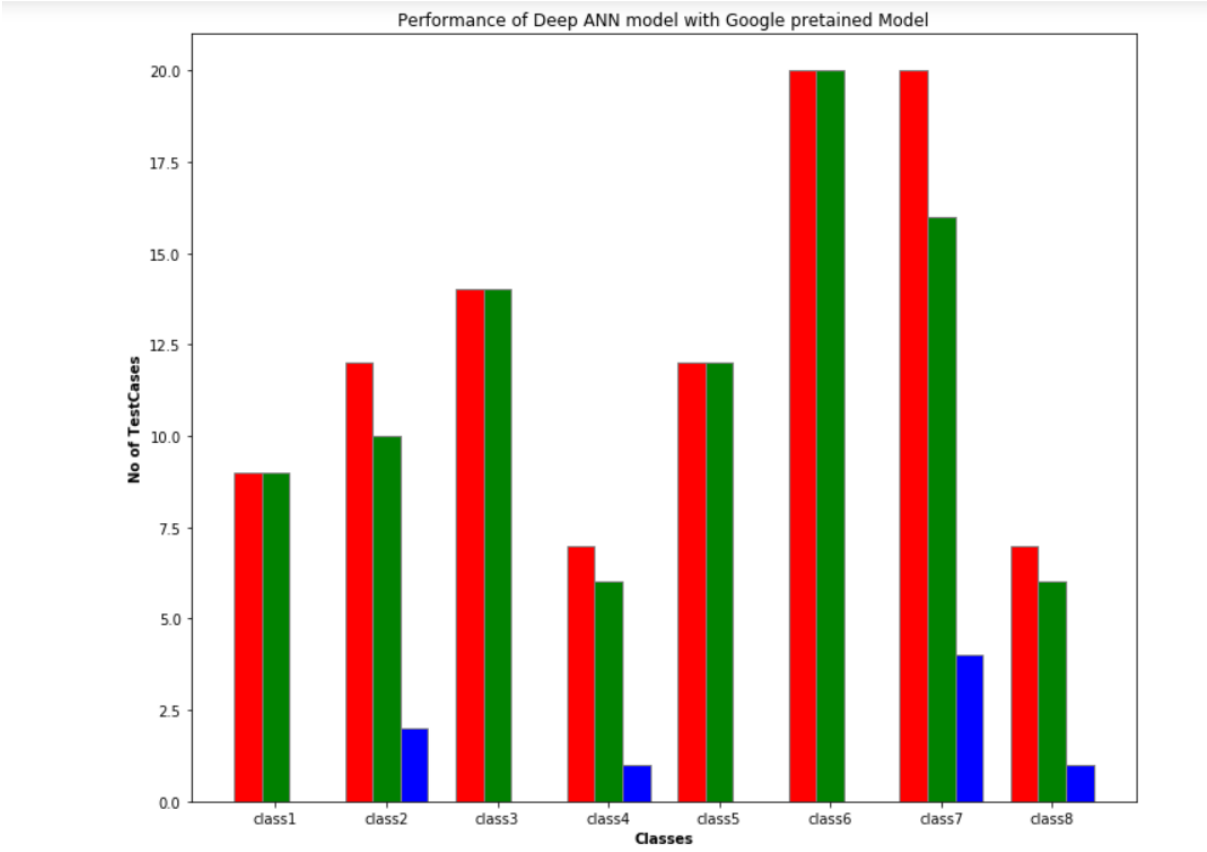
Classification report Comparison using Bar Graph

(Red bar-> total test cases, Green bar-> Correctly Classified test cases, Blue Bar-> Misclassified test cases)

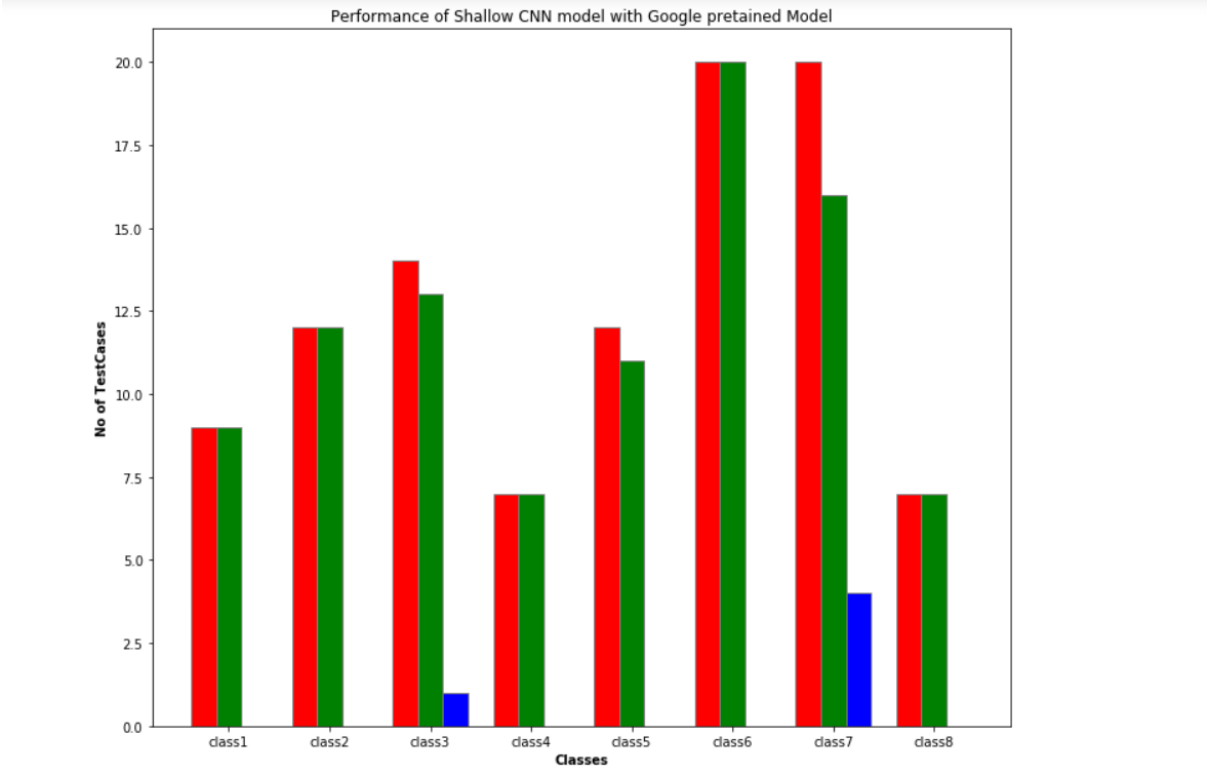
#model1 (CNN without Word2Vec pretrained Model)



#model2 (ANN with Word2Vec pretrained Model and deep network)



#model3 (CNN with Word2Vec pretrained Model and deep network)



Conclusion:

Clearly Model3 out performs every other model with 95% Accuracy as expected, as for NLP models generally Shallow network deep learning models perform better than the deep network Deep Learning models and also when working with strings as Word2Vec is suggested to work when dataset length is low.

Screenshots for the tabular data shown above

 # Model1

```
In [164]: from sklearn.metrics import confusion_matrix
          confusion_matrix(ytest, ypred)

Out[164]: array([[ 9,  0,  0,  0,  0,  0,  0,  0],
                 [ 3,  9,  0,  0,  0,  0,  0,  0],
                 [ 0,  0, 14,  0,  0,  0,  0,  0],
                 [ 0,  0,  0,  7,  0,  0,  0,  0],
                 [ 0,  0,  0,  0, 11,  0,  1,  0],
                 [ 0,  0,  0,  0,  0, 20,  0,  0],
                 [ 0,  0,  1,  0,  0,  0, 15,  4],
                 [ 0,  0,  0,  0,  0,  0,  3,  4]], dtype=int64)
```

```
In [165]: from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score
          precision_score(ytest, ypred, average='weighted')

Out[165]: 0.8921400034740316
```

```
In [167]: accuracy_score(ytest, ypred)

Out[167]: 0.8811881188118812
```

```
In [168]: from sklearn.metrics import f1_score
          f1_score(ytest, ypred, average='weighted')

Out[168]: 0.8823113957111316
```

#Model2

```
In [132]: from sklearn.metrics import confusion_matrix
          confusion_matrix(ytest, ypred)

Out[132]: array([[ 9,  0,  0,  0,  0,  0,  0,  0],
                 [ 2, 10,  0,  0,  0,  0,  0,  0],
                 [ 0,  0, 14,  0,  0,  0,  0,  0],
                 [ 0,  0,  0,  6,  0,  0,  1,  0],
                 [ 0,  0,  0,  0, 12,  0,  0,  0],
                 [ 0,  0,  0,  0,  0, 20,  0,  0],
                 [ 0,  0,  1,  0,  0,  0, 16,  3],
                 [ 0,  0,  0,  0,  0,  0,  1,  6]], dtype=int64)
```

```
In [133]: from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_scor
          precision_score(ytest, ypred, average='weighted')

Out[133]: 0.9294529452945294
```

In [137]:
```
accuracy_score(ytest, ypred)
```

Out[137]:
```
0.9207920792079208
```

In [136]:
```
from sklearn.metrics import f1_score
f1_score(ytest, ypred, average='weighted')
```

Out[136]:
```
0.9215839209217587
```

#Model3

In [152]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(ytest, ypred)
```

Out[152]:
```
array([[ 9,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 12,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 14,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  7,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 12,  0,  0,  0],
       [ 0,  0,  0,  0,  0, 20,  0,  0],
       [ 0,  0,  1,  0,  0,  0, 15,  4],
       [ 0,  0,  0,  0,  0,  0,  0,  7]], dtype=int64)
```

In [153]:
```
from sklearn.metrics import f1_score
f1_score(ytest, ypred, average='weighted')
```

Out[153]:
```
0.9515301283822964
```

In [154]:
```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_score
precision_score(ytest, ypred, average='weighted')
```

Out[154]:
```
0.9655565556555655
```

In [155]:
```
recall_score(ytest, ypred,average='weighted')
```

Out[155]:
```
0.9504950495049505
```

In [156]:
```
accuracy_score(ytest, ypred)
```

Out[156]:
```
0.9504950495049505
```