Final Project

CS 634

Dec 4, 2021

By: Rati Patel
Professor: Yasser Abduallah

Index

Introduction:

Machine learning and deep learning are both crucial to the data scientist job, so for this project, I chose option 1, which was to set up some algorithms against a data set of our choosing. While in the project proposal I said I would use Random Forest, SVM, and GRU for the deep learning algorithm, I found that in the end, LSTM was easier for me to work with, so I went with that instead.

The main computer I coded on has an AMD Ryzen 5 3600 3.6 GHz 6-Core Processor, running at 64 bits, 16 GB dual channel ram, and nearly 2 TB's of space.

The data set I chose was from here. Specifically, it's only their red wine data, as I thought it was easier to work with only one. Additionally, my choice in data comes from ease. Since this project is about the algorithms and understanding how those run, I figured it'd probably be better for me.

The attributes for this dataset is as follows:

1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulfates
11 - alcohol
12 - quality (score between 0 and 10) (this is the y value we're looking for)

Libraries Used

Pandas

Data frame, lets us import the data cleanly, as well as read and edit the data as needed.

NumPy

Lets us perform equations and other more advanced calculations more easily

Scikit/ SKLearn

Contains a multitude of machine learning methods, such as Random Forest Classifier, SVM (as Standard Vector Classifier), confusion matrix, splitting the data into train and test, and others.

Matplotlib

visualization / plot graphic builder

TensorFlow with Keras

Allows us to code and use Deep Learning methods.

Preliminary

Data Cleaning and Transforming

This data set was not missing any data, and for the most part, contained float or int data. No cleaning or transformation was needed in this case. The Data did require sep=”;” rather than the default, but that was really the only thing special about this data set, I’d say

Preprocessing

Preprocessing came in the form of simply defining our y (the wine quality) and our X (everything else)

```
In [25]: X= winedf.drop('quality', axis=1)
         y = winedf['quality']
```

I ran a KFold Method, to make sure the data was split into 10 (approximately) even slices.

```
In [28]: k = 10
         kf = KFold(n_splits=k, random_state=None)
         fc = 0
         ind = ['FP','FN','TP','TN','Positive','Negative','TPR','TNR','FPR','FNR','Precision','F1','Accuracy','Error','BACC','TSS','HS

         rf_df = pd.DataFrame(index=ind)
         svm_df = pd.DataFrame(index=ind)
         gru_df = pd.DataFrame(index=ind)
```

Splitting occurred within a for loop, to find the statistics matrix of each fold, and then the averages of all those folds.

Classification Algorithms:

Random Forest and SVM

Both Random Forest and SVM happen in tandem,

```
In [26]: from sklearn.ensemble import RandomForestClassifier
         classifier = RandomForestClassifier(n_estimators=50, random_state=8)
```

```
In [27]: from sklearn.svm import SVC
         model_svm = SVC()
```

```
In [29]: fc = 0
         for train_ind , test_ind in kf.split(X):
             fc=fc+1
             cn= 'fold '+str(fc)
             X_train , X_test = X.iloc[train_ind,:],X.iloc[test_ind,:]
             y_train , y_test = y[train_ind] , y[test_ind]

             #Random Forest
             classifier.fit(X_train,y_train)
             RF_pred = classifier.predict(X_test)

             Rf_cal = calc(y_test, RF_pred)
             rf_df[cn]=Rf_cal


           # SVM
             model_svm.fit(X_train, y_train)
             SVM_pred = model_svm.predict(X_test)

             svm_cal = calc(y_test, SVM_pred)
             svm_df[cn]=svm_cal


         rf_df['Average']=aveCalc(rf_df)
         svm_df['Average']=aveCalc(svm_df)
```

The calc algorithm looks like this

```
In [23]: def calc(labs, pred):
    cm = confusion_matrix(labs,pred)
    fp = int((cm.sum(axis=0) - np.diag(cm)).sum() )
    fn = int((cm.sum(axis=1) - np.diag(cm)).sum() )
    tp = int(np.diag(cm).sum())
    tn = int(abs(((cm.ravel().sum())*(cm.shape[1])) - (fp + fn + tp)))
    posi = tp + fn
    negi = tn +fp
    tpr = tp/posi
    tnr = tn/negi
    fpr= fp/negi
    fnr = fn / posi
    preci = tp/(tp+fp)
    f1 = (2 *tp)/(2 * tp + fp + fn)
    acc = (tp+tn)/(posi+negi)
    err = (fp+fn)/(posi + negi)
    bacc = (tpr+tnr)/2
    tss = (tp/(tp+fn))-(fp/(fp+tn))
    hss = (2*((tp*tn)-(fp*fn))/((tp+fn)*(fn+tn)+(tp+fp)*(fp+tn)))
    indval = [fp,fn,tp,tn,posi,negi,tpr,tnr,fpr,fnr,preci,f1,acc,err,bacc,tss,hss]
    return indval
```

For the False and True Positives and Negatives, I summed all the classes. So the numbers that return should be that of all false positives, false negatives, true positives, and true negatives for each fold, assuming I've done my math correctly.

Random Forest Results

Random Forest Results

```
In [30]: rf_df
```

Out[30]:

	fold 1	fold 2	fold 3	fold 4	fold 5	fold 6	fold 7	fold 8	fold 9	fold 10	Average
FP	57.000000	68.000000	79.000000	69.00000	58.000000	80.000000	70.000000	62.000000	57.00000	70.000000	67.000000
FN	57.000000	68.000000	79.000000	69.00000	58.000000	80.000000	70.000000	62.000000	57.00000	70.000000	67.000000
TP	103.000000	92.000000	81.000000	91.00000	102.000000	80.000000	90.000000	98.000000	103.00000	89.000000	92.900000
TN	583.000000	572.000000	721.000000	731.00000	582.000000	720.000000	570.000000	578.000000	743.00000	725.000000	652.500000
Positive	160.000000	160.000000	160.000000	160.00000	160.000000	160.000000	160.000000	160.000000	160.00000	159.000000	159.900000
Negative	640.000000	640.000000	800.000000	800.00000	640.000000	800.000000	640.000000	640.000000	800.00000	795.000000	719.500000
TPR	0.643750	0.575000	0.506250	0.56875	0.637500	0.500000	0.562500	0.612500	0.64375	0.559748	0.580975
TNR	0.910937	0.893750	0.901250	0.91375	0.909375	0.900000	0.890625	0.903125	0.92875	0.911950	0.906351
FPR	0.089063	0.106250	0.098750	0.08625	0.090625	0.100000	0.109375	0.096875	0.07125	0.088050	0.093649
FNR	0.356250	0.425000	0.493750	0.43125	0.362500	0.500000	0.437500	0.387500	0.35625	0.440252	0.419025
Precision	0.643750	0.575000	0.506250	0.56875	0.637500	0.500000	0.562500	0.612500	0.64375	0.559748	0.580975
F1	0.643750	0.575000	0.506250	0.56875	0.637500	0.500000	0.562500	0.612500	0.64375	0.559748	0.580975
Accuracy	0.857500	0.830000	0.835417	0.85625	0.855000	0.833333	0.825000	0.845000	0.88125	0.853249	0.847200
Error	0.142500	0.170000	0.164583	0.14375	0.145000	0.166667	0.175000	0.155000	0.11875	0.146751	0.152800
BACC	0.777344	0.734375	0.703750	0.74125	0.773438	0.700000	0.726562	0.757812	0.78625	0.735849	0.743663
TSS	0.554688	0.468750	0.407500	0.48250	0.546875	0.400000	0.453125	0.515625	0.57250	0.471698	0.487326
HSS	0.554688	0.468750	0.407500	0.48250	0.546875	0.400000	0.453125	0.515625	0.57250	0.471698	0.487326

SVM Results

In [31]:
```
svm_df
```

Out[31]:

	fold 1	fold 2	fold 3	fold 4	fold 5	fold 6	fold 7	fold 8	fold 9	fold 10	Average
FP	75.000000	69.000000	81.00000	82.000000	81.000000	101.000000	86.000000	82.000000	70.000000	83.000000	81.000000
FN	75.000000	69.000000	81.00000	82.000000	81.000000	101.000000	86.000000	82.000000	70.000000	83.000000	81.000000
TP	85.000000	91.000000	79.00000	78.000000	79.000000	59.000000	74.000000	78.000000	90.000000	76.000000	78.900000
TN	405.000000	571.000000	719.00000	718.000000	559.000000	699.000000	394.000000	558.000000	730.000000	712.000000	606.500000
Positive	160.000000	160.000000	160.00000	160.000000	160.000000	160.000000	160.000000	160.000000	160.000000	159.000000	159.900000
Negative	480.000000	640.000000	800.00000	800.000000	640.000000	800.000000	480.000000	640.000000	800.000000	795.000000	687.500000
TPR	0.531250	0.568750	0.49375	0.487500	0.493750	0.368750	0.462500	0.487500	0.562500	0.477987	0.493424
TNR	0.843750	0.892188	0.89875	0.897500	0.873437	0.873750	0.820833	0.871875	0.912500	0.895597	0.878018
FPR	0.156250	0.107813	0.10125	0.102500	0.126562	0.126250	0.179167	0.128125	0.087500	0.104403	0.121982
FNR	0.468750	0.431250	0.50625	0.512500	0.506250	0.631250	0.537500	0.512500	0.437500	0.522013	0.506576
Precision	0.531250	0.568750	0.49375	0.487500	0.493750	0.368750	0.462500	0.487500	0.562500	0.477987	0.493424
F1	0.531250	0.568750	0.49375	0.487500	0.493750	0.368750	0.462500	0.487500	0.562500	0.477987	0.493424
Accuracy	0.765625	0.827500	0.83125	0.829167	0.797500	0.789583	0.731250	0.795000	0.854167	0.825996	0.804704
Error	0.234375	0.172500	0.16875	0.170833	0.202500	0.210417	0.268750	0.205000	0.145833	0.174004	0.195296
BACC	0.687500	0.730469	0.69625	0.692500	0.683594	0.621250	0.641667	0.679688	0.737500	0.686792	0.685721
TSS	0.375000	0.460938	0.39250	0.385000	0.367188	0.242500	0.283333	0.359375	0.475000	0.373585	0.371442
HSS	0.375000	0.460938	0.39250	0.385000	0.367188	0.242500	0.283333	0.359375	0.475000	0.373585	0.371442

LSTM:

Here is the set up for my Deep Learning implementation.

LSTM

```
In [32]: from sklearn.preprocessing import MinMaxScaler
         from keras.models import Sequential
         from keras.layers import Dense, LSTM
         from keras import metrics

In [ ]:

In [ ]:

In [42]: lstm_mod = Sequential()
         lstm_mod.add(Dense(12, activation ='softmax', input_shape =(11, )))
         lstm_mod.add(Dense(9, activation ='softmax'))
         lstm_mod.add(Dense(1, activation ='sigmoid'))
         lstm_mod.output_shape
         lstm_mod.summary()
         lstm_mod.get_config()

         # List all weight tensors
         lstm_mod.get_weights()
         lstm_mod.compile(loss ='binary_crossentropy',
           optimizer ='adamax', metrics = [metrics.categorical_accuracy])
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 12)                144
_________________________________________________________________
dense_16 (Dense)             (None, 9)                 117
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 10
=================================================================
Total params: 271
Trainable params: 271
Non-trainable params: 0
_________________________________________________________________
```

As I will state next, I can't say I really understand what it's done here. I've run it a few times to try and get an understanding and… I'm rather confused.

Problems

I struggled a bit with getting the matrix to return a proper number. The

To be honest, I don't really understand Deep learning. I figured out how to set up the functions to condense the arrays as needed, but I don't actually understand how to get the confusion matrix. So I got results, I think, but I don't know what to do with them. I apologize for their length