test_trained_model

March 9, 2019

#!/usr/bin/env python3 # -*- coding: utf-8 -*-

Created on Sat Mar 9 20:30:44 2019

@author: kareem @Date: 09.03.2019 @Title: Last script in my project to load trained model and evaluate it against a test set.

0.1 Loading Model weights and architecture

Load the previously trained keras model and adjusted weights, and compile it with best found parameters.

```
In [1]: ### Loading Model weights and architecture ####
        import keras
        from keras.models import model_from_json
        from keras.metrics import binary_accuracy
        from keras.optimizers import SGD
        ## load json and create model
        jf = open('analysis_best_model/best_model.json', 'r')
        best_model = model_from_json(jf.read())
        jf.close()
        ## load weights into new model
        best_model.load_weights("analysis_best_model/best_model.h5")
        ## Compile the model after loading
        best_model.compile(optimizer=SGD(lr=0.1), loss='binary_crossentropy', metrics=[binary_a
```

```
Using TensorFlow backend.
```

0.2 Read Test data

In one loop read the rows from the csv file and read the SMILES into molecules, then calculate the Morgan Fingerpritns as a bit vector and store them in `fingerprints` list. Also, append the label on the same row to list `y_true`.

```
In [3]: #### Read Test data ####
        import csv
        from rdkit import Chem
        from rdkit.Chem import AllChem
        fingerprints = []
```

```
y_true = []
info={}
with open('data/test.csv', 'r') as f:
    reader = csv.reader(f)
    i=0
    for row in reader:
        if i==0:
            i+=1
            continue
        ## read molecule from SMILES
        m = Chem.MolFromSmiles(row[1])
        ## Calculate fingerprint accordingly
        fp = AllChem.GetMorganFingerprintAsBitVect(m, 3, nBits=2048, bitInfo=info)
        ## Append fp into fingerprints
        fingerprints.append(fp)
        y_true.append(int(row[2]))
```

0.3 Standard Scaling

Apply Standard Scaling for ease of training and fast computation

In [4]:
```
#### Standard Scaling ####
from sklearn.preprocessing import StandardScaler
import numpy as np
#Scale fingerprints to unit variance and zero mean
scaler = StandardScaler()
X_test = scaler.fit_transform(np.array(fingerprints))
y_true = np.array(y_true)
```

```
/anaconda3/lib/python3.5/site-packages/sklearn/utils/validation.py:475: DataConversionWarning:
  warnings.warn(msg, DataConversionWarning)
```

In [8]:
```
from itertools import product
from matplotlib import pyplot as plt
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')
```

```
        print(cm)

        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title)
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation=45)
        plt.yticks(tick_marks, classes)

        fmt = '.2f' if normalize else 'd'
        thresh = cm.max() / 2.
        for i, j in product(range(cm.shape[0]), range(cm.shape[1])):
            plt.text(j, i, format(cm[i, j], fmt),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")

        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.tight_layout()
```

```
In [9]: def plot_roc_curve(fpr, tpr, auc):
            plt.figure()
            plt.plot(fpr, tpr, color='brown',
                     lw=2, label='ROC curve (AUC = %0.2f)' % auc)
            plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
            plt.xlim([0.0, 1.0])
            plt.ylim([0.0, 1.0])
            plt.xlabel('False Positive Rate')
            plt.ylabel('True Positive Rate')
            plt.title('Receiver Operating Characteristic')
            plt.legend(loc="lower right")
            plt.savefig('Best_model_ROC')
            plt.show()
```

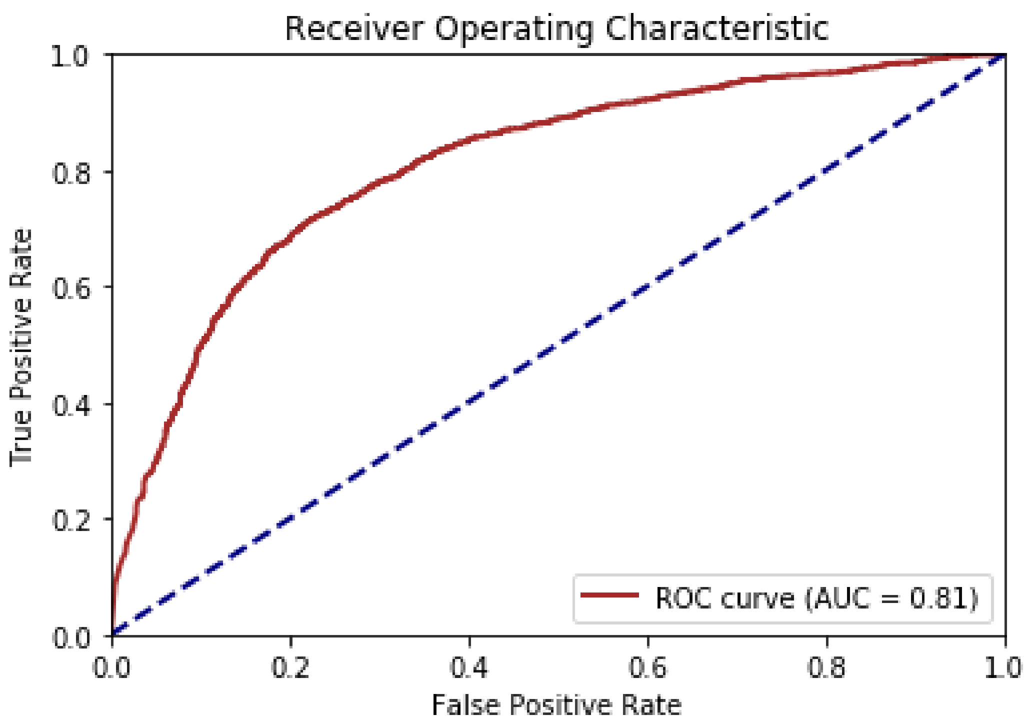
0.4 Predicting on the Test Set

In order to run the following analysis you should load the two functions above: * first to plot the Confusion Matrix * Second to plot the ROC curve

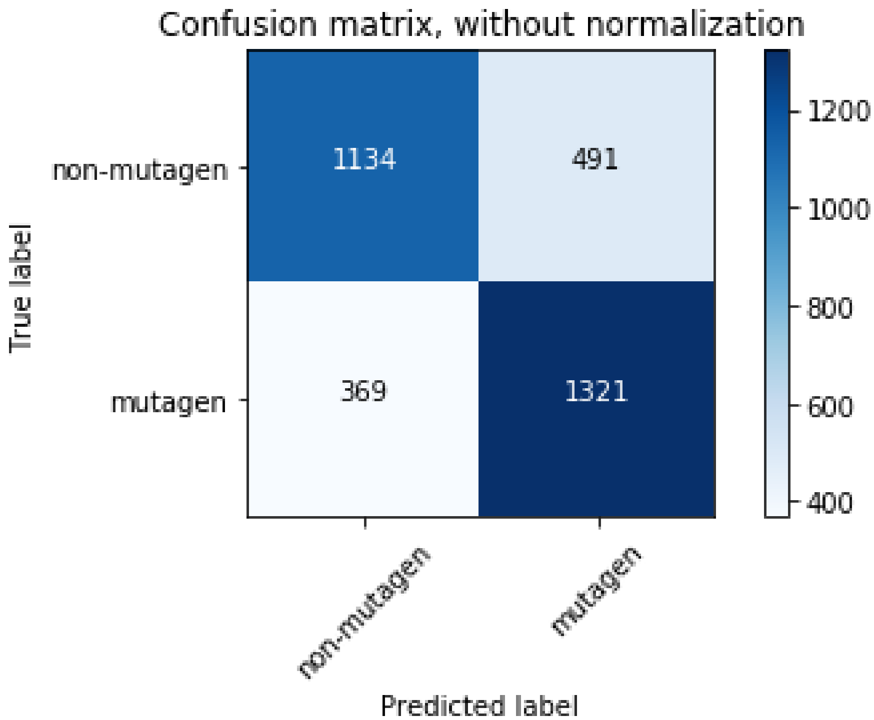
```
In [10]: #### Predicting on the Test Set ####
         y_prob = best_model.predict(X_test)
         y_pred = best_model.predict_classes(X_test)

         ### Evaluating Model's predictions ####
         from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix
         auc = roc_auc_score(y_true, y_prob)
         fpr, tpr, thresholds = roc_curve(y_true, y_prob)
```

```python
plot_roc_curve(fpr, tpr, auc) # load this function!
## Construct and plot confusion matrix
cm = confusion_matrix(y_true, y_pred)
class_names = np.array(['non-mutagen', 'mutagen'])
plot_confusion_matrix(cm, classes=class_names, # load this function!
                      title='Confusion matrix, without normalization')
```




```
Confusion matrix, without normalization
[[1134  491]
 [ 369 1321]]
```


Confusion matrix, without normalization
non-mutagen
mutagen
True label
1134
491
369
1321
1200
1000
800
600
400
non-mutagen
mutagen
Predicted label

In [ ]: