Project_BinaryRelevance

December 15, 2020

[1]:
```
#import data set
import pandas as pd
import numpy as np
import re
import string

test = pd.read_csv('data/NLP_on_Research_Articles/archive/test.csv')
train = pd.read_csv('data/NLP_on_Research_Articles/archive/train.csv')

print(train.shape)
print(test.shape)
```

```
(20972, 9)
(8989, 3)
```

[2]:
```
#deal with the column names
train.columns = train.columns.str.strip().str.lower().str.replace(' ', '_').str.
 ↪replace('(', '').str.replace(')', '')
print('Train Data shape: ', train.shape)
```

```
Train Data shape:  (20972, 9)
```

[3]:
```
def remove_pattern(text, pattern):
    r = re.findall(pattern, text)
    for i in r:
        text = re.sub(i, "", text)
    return text
```

[4]:
```
#clean data
for column in ['title', 'abstract']:
    train[column] = np.vectorize(remove_pattern)(train[column], "@[\w]*")
    train[column] = np.vectorize(remove_pattern)(train[column], "#[\w]*")
    train[column] = np.vectorize(remove_pattern)(train[column], '[0-9]')
    train[column] = train[column].str.replace("[^a-zA-Z#]", " ")
    train[column] = train[column].apply(lambda x: ' '.join([i for i in x.
 ↪split() if len(i) > 3]))

train['description'] = train['title'] + " " + train['abstract']
```

```
train['description'] = train['description'].str.lower()
```

[5]:
```
#TfidfVectorizer the documents
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words = 'english')
vectors_train = vectorizer.fit_transform(train["description"])
print(vectors_train.shape)
label_train = np.asarray(train[train.columns[3:9]].values)
print(label_train.shape)
```

```
(20972, 46333)
(20972, 6)
```

[6]:
```
#split train data to 2:1
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(vectors_train, label_train,␣
↪test_size=0.33, random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(14051, 46333)
(6921, 46333)
(14051, 6)
(6921, 6)
```

0.1 1.KNN

[ ]:
```
#Use knn classifier directly, here the size of y_train is (14051, 6), the size␣
↪of y_test is (6921, 6)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
n = list(range(1,13))
test_scores = []
for k in n:
    model = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
    model.fit(x_train,y_train)
    predicted= model.predict(x_test)
    knn_scores = accuracy_score(y_test, predicted)
    print(knn_scores)
    test_scores.append(knn_scores)
print(test_scores)
```

[ ]:
```
import matplotlib.pyplot as plt

#plot
```

```
plt.plot(n, test_scores, label='plain_knn_error', marker='*', markersize=12,␣
 ↪color='tab:orange')
plt.legend(loc='upper center')
plt.xlabel("k")
plt.ylabel("test score")
plt.show()
plt.savefig('fig1_1.pdf')
```

[7]:
```
#Calculate test accuracy by fractional values
def accuracy_fraction(y_test, predicted):
    test_size = y_test.shape[0]
    frac_values = []
    for i in range(0,test_size):
        single_frac = accuracy_score(y_test[i,], predicted[i,])
        frac_values.append(single_frac)
    test_scores = sum(frac_values)/test_size
    return test_scores
```

[29]:
```
test_scores = accuracy_fraction(y_test, predicted)
print(test_scores)
```

```
0.905336415739529
```

[34]:
```
#Calculate average accuracy
#predicted = predicted.toarray()
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print(sum(accuracy_label)/6)
```

```
0.9053364157395367
```

[ ]:
```
label_size = y_test.shape[1]
frac_values = []
for i in range(0,test_size):
    single_frac = accuracy_score(y_test[i,], predicted[i,])
    frac_values.append(single_frac)
test_scores = sum(frac_values)/test_size
```

[8]:
```
#Use knn classifier directly, here the size of y_train is (14051, 6), the size␣
 ↪of y_test is (6921, 6)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

model = KNeighborsClassifier(n_neighbors=9, metric='euclidean')
model.fit(x_train,y_train)
predicted= model.predict(x_test)
```

```
knn_scores = accuracy_score(y_test, predicted)
```

[10]:
```
predicted
```

[10]:
```
array([[1, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0],
       …,
       [0, 0, 1, 0, 0, 0],
       [0, 1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0]])
```

[9]:
```
print(knn_scores)
```

```
0.603958965467418
```

0.2 2.NLC

[ ]:
```
```

0.3 3. LDA

0.4 95% PCA

[ ]:
```
#PCA
#Perform PCA, find k to preserve 95% variation
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(vectors_train.toarray())
#explained variances cumulatively
cumVar = np.cumsum(pca.explained_variance_ratio_, dtype=float)
#find k
col_num = vectors_train.shape[1]
for k in range(0,col_num):
    if cumVar[k] >= 0.95:
        break
print(k)
#PCA with 95% variation preserved
pca = PCA(n_components=k+1)
pca.fit(vectors_train.toarray())
x_train_pca = pca.transform(vectors_train.toarray())
print(x_train_pca.shape)
```

[ ]:
```
import pandas as pd
pd.DataFrame(x_train_pca).to_csv('pca_train.csv')
```

[ ]:
```
#PCA split
#split train data to 2:1
from sklearn.model_selection import train_test_split
pca_x_train, pca_x_test, pca_y_train, pca_y_test =␣
↪train_test_split(x_train_pca, label_train,
                                                    test_size=0.
↪33, random_state=42)
print(pca_x_train.shape)
print(pca_x_test.shape)
print(pca_y_train.shape)
print(pca_y_test.shape)
```

[ ]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.metrics import accuracy_score
import time

start_time = time.time()
test_error_LDA = []

lda = BinaryRelevance(LinearDiscriminantAnalysis())
predicted = lda.fit(pca_x_train,pca_y_train).predict(pca_x_test)
LDA_scores = accuracy_score(pca_y_test, predicted)
print("--- %s seconds ---" % round(time.time() - start_time,2))

test_error_LDA.append(1-LDA_scores)
print(LDA_scores)
print(test_error_LDA)
print("f1 score: ", f1_score(pca_y_test, predicted, average='micro'))
```

0.5 4.QDA

[ ]:
```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

import time

start_time = time.time()

test_error_QDA = []

qda = BinaryRelevance(QuadraticDiscriminantAnalysis())
predicted = qda.fit(pca_x_train,pca_y_train).predict(pca_x_test)
QDA_scores = accuracy_score(pca_y_test, predicted)
```

```
print("--- %s seconds ---" % round(time.time() - start_time,2))

test_error_QDA.append(1-QDA_scores)
print(QDA_scores)
print(test_error_QDA)
print("f1 score: ", f1_score(pca_y_test, predicted, average='micro'))
```

0.6 5. MultinomialNB

[1]:
```
# using binary relevance
from skmultilearn.problem_transform import BinaryRelevance
#from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
import time

start_time = time.time()

# initialize binary relevance multi-label classifier
classifier = BinaryRelevance(MultinomialNB())

# train
classifier.fit(x_train, y_train)

# predict
predicted = classifier.predict(x_test)

test_scores = accuracy_score(y_test, predicted)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",test_scores)
#predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

[11]:
```
#Calculate average accuracy
predicted = predicted.toarray()
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print(sum(accuracy_label)/6)
```

```
0.8968597986803449
```

[ ]:
```

```

0.7 6. GaussianNB

[ ]:
```
# using binary relevance
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.naive_bayes import GaussianNB
import time

start_time = time.time()

# initialize binary relevance multi-label classifier
# with a gaussian naive bayes base classifier
classifier = BinaryRelevance(GaussianNB())

# train
classifier.fit(x_train, y_train)

# predict
predicted = classifier.predict(x_test)

test_scores = accuracy_score(y_test, predicted)

print("--- %s seconds ---" % round(time.time() - start_time,2))
print(test_scores)
print("f1 score: ", f1_score(y_test, predicted, average='micro'))
```

0.8 7. LR

[ ]:
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

import numpy as np
import time

#one vs rest
LR_ovr = []
for i in list(range(-6,6)):
    print(i)
    start_time = time.time()
    ovr_clf = OneVsRestClassifier(LogisticRegression(C=2**i,␣
 ↪solver='newton-cg', max_iter=4000)).fit(x_train, y_train)
    predicted = ovr_clf.predict(x_test) # prediction of labels
    ovr_scores = accuracy_score(y_test, predicted)
    print("--- %s seconds ---" % round(time.time() - start_time,2))
    print("ovr_scores", ovr_scores)
    print("f1 score: ", f1_score(y_test, predicted, average='micro'))
```

```
    LR_ovr.append(ovr_scores)

print(LR_ovr)
```

[ ]:
```
#cross validation for LR
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
import numpy as np
import time

#cross validation accuracy for each k
cv_scores = []

#one vs rest
for i in list(range(-6,6)):
    print(i)
    start_time = time.time()
    ovr_clf = OneVsRestClassifier(LogisticRegression(C=2**i,␣
 ↪solver='newton-cg', max_iter=4000))
    scores = cross_val_score(ovr_clf, x_train, y_train, cv=6,␣
 ↪scoring='accuracy')
    cv_scores.append(scores.mean())
    print("--- %s seconds ---" % round(time.time() - start_time,2))
    print("ovr_scores", cv_scores)
    #print("f1 score: ", f1_score(y_test, predicted, average='micro'))


#6-fold cross validation to the training set
```

0.9 8. SVM linear

[ ]:
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import numpy as np
import time

svc_ovr = []
for i in list(range(-6,6)):
    print(i)
    start_time = time.time()
    ovr_clf = OneVsRestClassifier(LinearSVC(random_state=0, C=2**i,␣
 ↪penalty='l2')).fit(x_train, y_train)
```

```
    predicted = ovr_clf.predict(x_test)
    test_scores = accuracy_score(y_test, predicted)
    print("--- %s seconds ---" % round(time.time() - start_time,2))
    print("Accuracy: ", test_scores)
    print("f1 score: ", f1_score(y_test, predicted, average='micro'))
    svc_ovr.append(test_scores)
print(svc_ovr)
```

0.10 9. SVM poly

[ ]:
```
from sklearn import svm
from sklearn.metrics import accuracy_score
import numpy as np
import time

svc_poly = []
num_labels = y_test.shape[1]
predicted = np.zeros((y_test.shape[0], num_labels))

for i in list(range(1,2)):
    start_time = time.time()
    print(i)
    for j in range(0,num_labels):
        ovr_poly = svm.SVC(kernel='poly', degree=3, gamma='auto',C=2**i).
 ↪fit(x_train, y_train[:,j])
        predicted[:,j] = ovr_poly.predict(x_test)
    test_scores = accuracy_score(y_test, predicted)
    print("--- %s seconds ---" % round(time.time() - start_time,2))
    print("ovr_scores", test_scores)
    svc_poly.append(test_scores)

print(svc_poly)
print("Done!")
```

0.11 10. SVM Gaussian kernel

[ ]:
```
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

import numpy as np
import time

#C: number of labels
num_labels = y_test.shape[1]
predicted = np.zeros((y_test.shape[0], num_labels))
```

```
for i in list(range(-6,6)):
    start_time = time.time()
    print(i)
    for j in range(0,num_labels):
        ovr_poly = svm.SVC(kernel='rbf',gamma='scale',C=2**i).fit(x_train,␣
 ↪y_train[:,j])
        predicted[:,j] = ovr_poly.predict(x_test)
    test_scores = accuracy_score(y_test, predicted)
    print("--- %s seconds ---" % round(time.time() - start_time,2))
    print("Gaussian kernel SVM score:", test_scores)
    print("f1 score: ", f1_score(y_test, predicted, average='micro'))


print("Done!")
```

0.12 11. RandomForest_Decision tree

[ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import time

tree_num = list(range(5,501,5))

rf_test_scores = []
rf_oob_scores = []

for i in tree_num:
    print(i)
    start_time = time.time()
    clf = RandomForestClassifier(n_estimators=i, bootstrap=True,␣
 ↪max_features='sqrt',
                                 oob_score=True, random_state=0).fit(x_train,␣
 ↪y_train)
    predicted = clf.predict(x_test)
    test_scores = accuracy_score(y_test, predicted)
    print("--- %s seconds ---" % round(time.time() - start_time,2))
    print(test_scores)
    print(clf.oob_score_)
    rf_test_scores.append(test_scores)
    rf_oob_scores.append(clf.oob_score_)

print("Done!")
print("rf_test_scores", rf_test_scores)
print("rf_oob_scores", rf_oob_scores)
```

[ ]:
```
#plot
import matplotlib.pyplot as plt

n = list(range(5,501,5))
rf_oob_errors = [1-x for x in rf_oob_scores]
rf_test_errors = [1-x for x in rf_test_scores]
plt.plot(n, rf_oob_errors, label='oob error', marker='o', markersize=4,␣
↪color='royalblue')
plt.plot(n, rf_test_errors, label='test error', marker='*', markersize=4,␣
↪color='tab:orange')
plt.xticks(np.arange(0, 501, 50))
plt.legend(loc='upper right')
plt.xlabel("Number of trees")
plt.ylabel("errors")
#plt.savefig('fig6_1.pdf')
plt.show()
```

0.13 12. Bagging

[ ]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import time

start_time = time.time()
#number of labels
num_labels = y_test.shape[1]
predicted = np.zeros((y_test.shape[0], num_labels))


for i in range(0,num_labels):
    clf = BaggingClassifier(DecisionTreeClassifier(max_depth=100),␣
↪n_estimators=200, bootstrap=True,
                            oob_score=True, random_state=0).fit(x_train,␣
↪y_train[:,i])
    predicted[:,i] = clf.predict(x_test)

test_scores = accuracy_score(y_test, predicted)
print("--- %s seconds ---" % round(time.time() - start_time,2))

print(test_scores)
print(clf.oob_score_)
```

0.14 13. Adaptive boosting

[ ]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score
import time

start_time = time.time()
#number of labels
num_labels = y_test.shape[1]
predicted = np.zeros((y_test.shape[0], num_labels))

for i in range(0,num_labels):
    clf = AdaBoostClassifier(DecisionTreeClassifier(max_depth=100),␣
 ↪n_estimators=200, random_state=0).fit(x_train, y_train[:,i])
    predicted[:,i] = clf.predict(x_test)

test_scores = accuracy_score(y_test, predicted)
print("--- %s seconds ---" % round(time.time() - start_time,2))

print(test_scores)
```

0.15 Take LinearSVM for example

[ ]:
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import numpy as np
import time

start_time = time.time()
ovr_clf = OneVsRestClassifier(LinearSVC(random_state=0, C=0.25, penalty='l2')).
 ↪fit(x_train, y_train)
predicted = ovr_clf.predict(x_test)
test_scores = accuracy_score(y_test, predicted)
print("--- %s seconds ---" % round(time.time() - start_time,2))

print("Accuracy: ", test_scores)
print("f1 score: ", f1_score(y_test, predicted, average='micro'))
```

[ ]:
```
for i in list(range(0,6)):
    print(i)
    print(accuracy_score(y_test[:,i],predicted[:,i]))
```

[ ]:
```
misclassification = (y_test == predicted).all(axis=1)
i, = np.where(misclassification == False)
j, = np.where(misclassification == True)
mis_y_test = y_test[i]
correct_y_test = y_test[j]
print(len(mis_y_test))
print(len(correct_y_test))
print(len(y_test))
```

[ ]:
```
#mis articles that only belong to 1 label.
x = np.array([[1, 0, 0, 0, 0, 0],
              [0, 1, 0, 0, 0, 0],
              [0, 0, 1, 0, 0, 0],
              [0, 0, 0, 1, 0, 0],
              [0, 0, 0, 0, 1, 0],
              [0, 0, 0, 0, 0, 1]])
c = x.shape[0]
mis_num = []
correct_num = []
train_num = []

for i in range(0,c):
    each_combine = np.repeat(([x[i,:]]), len(mis_y_test), axis=0)
    paired_mis_num = accuracy_score(mis_y_test, each_combine, False)
    mis_num.append(paired_mis_num)

    each_combine = np.repeat(([x[i,:]]), len(correct_y_test), axis=0)
    paired_correct_num = accuracy_score(correct_y_test, each_combine, False)
    correct_num.append(paired_correct_num)

    each_combine = np.repeat(([x[i,:]]), len(y_train), axis=0)
    paired_train_num = accuracy_score(y_train, each_combine, False)
    train_num.append(paired_train_num)

print(mis_num)
print(correct_num)
print(train_num)

#plot
import matplotlib.pyplot as plt
# set width of bar
barWidth = 0.25
# Set position of bar on X axis
r1 = np.arange(len(correct_num))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
```

```
# Make the plot
plt.bar(r1, train_num, color='royalblue', width=barWidth, edgecolor='white',␣
 ↪label='Train data')
plt.bar(r2, correct_num, color='green', width=barWidth, edgecolor='white',␣
 ↪label='Correctly classified')
plt.bar(r3, mis_num, color='red', width=barWidth, edgecolor='white',␣
 ↪label='Misclassified')

# Add xticks on the middle of the group bars
plt.title('One Label Articles (6)')
plt.xlabel('Label', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(correct_num))], ["1" ,"2" , "3",␣
 ↪"4" , "5" , "6"])

# Create legend & Show graphic
plt.legend()
plt.savefig('BR_project_1.pdf',bbox_inches='tight')
plt.show()
```

[ ]:
```
#mis articles that only belong to 2 label.
x = np.array([[1, 1, 0, 0, 0, 0],
              [1, 0, 1, 0, 0, 0],
              [1, 0, 0, 1, 0, 0],
              [1, 0, 0, 0, 1, 0],
              [1, 0, 0, 0, 0, 1],
              [0, 1, 1, 0, 0, 0],
              [0, 1, 0, 1, 0, 0],
              [0, 0, 1, 1, 0, 0],
              [0, 0, 0, 1, 1, 0],
              [0, 0, 0, 1, 0, 1],
              [0, 0, 0, 0, 1, 1]])

c = x.shape[0]
mis_num = []
correct_num = []
train_num = []

for i in range(0,c):
    each_combine = np.repeat(([x[i,:]]), len(mis_y_test), axis=0)
    paired_mis_num = accuracy_score(mis_y_test, each_combine, False)
    mis_num.append(paired_mis_num)

    each_combine = np.repeat(([x[i,:]]), len(correct_y_test), axis=0)
    paired_correct_num = accuracy_score(correct_y_test, each_combine, False)
    correct_num.append(paired_correct_num)

    each_combine = np.repeat(([x[i,:]]), len(y_train), axis=0)
```

```
    paired_train_num = accuracy_score(y_train, each_combine, False)
    train_num.append(paired_train_num)

print(mis_num)
print(correct_num)
print(train_num)

#plot
import matplotlib.pyplot as plt
# set width of bar
barWidth = 0.25
# Set position of bar on X axis
r1 = np.arange(len(correct_num))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

# Make the plot
plt.bar(r1, train_num, color='royalblue', width=barWidth, edgecolor='white',␣
 ↪label='Train data')
plt.bar(r2, correct_num, color='green', width=barWidth, edgecolor='white',␣
 ↪label='Correctly classified')
plt.bar(r3, mis_num, color='red', width=barWidth, edgecolor='white',␣
 ↪label='Misclassified')

# Add xticks on the middle of the group bars
target_labels = ["1+2" ,"1+3" , "1+4", "1+5" , "1+6" , "2+3", "2+4", "3+4",␣
 ↪"4+5",
                  "4+6", "5+6"]
plt.title('Two Label Articles (11)')
plt.xlabel('Labels', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(correct_num))], target_labels)

# Create legend & Show graphic
plt.legend()
plt.savefig('BR_project_2.pdf',bbox_inches='tight')
plt.show()
```

[ ]:
```
#mis articles that only belong to 3 label.
x = np.array([[1, 1, 1, 0, 0, 0],
              [1, 1, 0, 1, 0, 0],
              [0, 1, 1, 1, 0, 0],
              [0, 0, 1, 1, 0, 1],
              [1, 0, 1, 1, 0, 0],
              [1, 0, 0, 1, 1, 0],
              [1, 0, 0, 1, 0, 1]])
c = x.shape[0]
mis_num = []
```

```
correct_num = []
train_num = []

for i in range(0,c):
    each_combine = np.repeat(([x[i,:]]), len(mis_y_test), axis=0)
    paired_mis_num = accuracy_score(mis_y_test, each_combine, False)
    mis_num.append(paired_mis_num)

    each_combine = np.repeat(([x[i,:]]), len(correct_y_test), axis=0)
    paired_correct_num = accuracy_score(correct_y_test, each_combine, False)
    correct_num.append(paired_correct_num)

    each_combine = np.repeat(([x[i,:]]), len(y_train), axis=0)
    paired_train_num = accuracy_score(y_train, each_combine, False)
    train_num.append(paired_train_num)

print(mis_num)
print(correct_num)
print(train_num)

#plot
import matplotlib.pyplot as plt
# set width of bar
barWidth = 0.25
# Set position of bar on X axis
r1 = np.arange(len(correct_num))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

# Make the plot
plt.bar(r1, train_num, color='royalblue', width=barWidth, edgecolor='white',␣
 ↪label='Train data')
plt.bar(r2, correct_num, color='green', width=barWidth, edgecolor='white',␣
 ↪label='Correctly classified')
plt.bar(r3, mis_num, color='red', width=barWidth, edgecolor='white',␣
 ↪label='Misclassified')

# Add xticks on the middle of the group bars
target_labels = ["1+2+3" ,"1+2+4" , "2+3+4", "3+4+6" , "1+3+4" , "1+4+5",␣
 ↪"1+4+6"]
plt.title('Three Label Articles (7)')
plt.xlabel('Labels', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(correct_num))], target_labels)

# Create legend & Show graphic
plt.legend()
plt.savefig('BR_project_3.pdf',bbox_inches='tight')
```

```
plt.show()
```

[ ]:

[ ]:

[ ]:

Project_LabelPowerSet

December 15, 2020

[1]:
```
#import data set
import pandas as pd
import numpy as np
import re
import string

test = pd.read_csv('data/NLP_on_Research_Articles/archive/test.csv')
train = pd.read_csv('data/NLP_on_Research_Articles/archive/train.csv')

print(train.shape)
print(test.shape)
```

```
(20972, 9)
(8989, 3)
```

[2]:
```
#deal with the column names
train.columns = train.columns.str.strip().str.lower().str.replace(' ', '_').str.
 ↪replace('(', '').str.replace(')', '')
print('Train Data shape: ', train.shape)
```

```
Train Data shape:  (20972, 9)
```

[3]:
```
def remove_pattern(text, pattern):
    r = re.findall(pattern, text)
    for i in r:
        text = re.sub(i, "", text)
    return text
```

[4]:
```
#clean data
for column in ['title', 'abstract']:
    train[column] = np.vectorize(remove_pattern)(train[column], "@[\w]*")
    train[column] = np.vectorize(remove_pattern)(train[column], "#[\w]*")
    train[column] = np.vectorize(remove_pattern)(train[column], '[0-9]')
    train[column] = train[column].str.replace("[^a-zA-Z#]", " ")
    train[column] = train[column].apply(lambda x: ' '.join([i for i in x.
 ↪split() if len(i) > 3]))

train['description'] = train['title'] + " " + train['abstract']
```

```
train['description'] = train['description'].str.lower()
```

[5]:
```
#TfidfVectorizer the documents
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words = 'english')
vectors_train = vectorizer.fit_transform(train["description"])
print(vectors_train.shape)
label_train = np.asarray(train[train.columns[3:9]].values)
print(label_train.shape)
```

```
(20972, 46333)
(20972, 6)
```

[6]:
```
#split train data to 2:1
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(vectors_train, label_train,␣
↪test_size=0.33, random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(14051, 46333)
(6921, 46333)
(14051, 6)
(6921, 6)
```

[7]:
```
#Calculate test accuracy by fractional values
def accuracy_fraction(y_test, predicted):
    test_size = y_test.shape[0]
    frac_values = []
    for i in range(0,test_size):
        single_frac = accuracy_score(y_test[i,], predicted[i,])
        frac_values.append(single_frac)
    test_scores = sum(frac_values)/test_size
    return test_scores
```

[ ]:
```

```

0.1 1.LR

[14]:
```
from skmultilearn.problem_transform import LabelPowerset
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time
```

```
start_time = time.time()

lp_classifier = LabelPowerset(LogisticRegression(C=4, solver='newton-cg',␣
↪max_iter=4000))
lp_classifier.fit(x_train, y_train)
lp_predictions = lp_classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

print("Accuracy = ", accuracy_score(y_test, lp_predictions))
print("F1 score = ", f1_score(y_test, lp_predictions, average = "micro"))

lp_predictions = lp_predictions.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, lp_predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],lp_predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:469:
FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify
the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

```
--- 15199.68 seconds ---
Accuracy =  0.6799595434185811
F1 score =  0.8068849706129303
Fraction Accuracy =  0.9224582189471601
Average Accuracy =  0.9224582189471656
```

0.2 2.SVM linear

[10]:
```
from skmultilearn.problem_transform import LabelPowerset
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time

start_time = time.time()

lp_classifier = LabelPowerset(LinearSVC(random_state=0, C=0.25, penalty='l2'))
lp_classifier.fit(x_train, y_train)
lp_predictions = lp_classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

print("Accuracy = ", accuracy_score(y_test, lp_predictions))
print("F1 score = ", f1_score(y_test, lp_predictions, average = "micro"))

lp_predictions = lp_predictions.toarray()
```

```
print("Fraction Accuracy = ", accuracy_fraction(y_test, lp_predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],lp_predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
--- 15.49 seconds ---
Accuracy =  0.6828492992342147
F1 score =  0.8065502969228
Fraction Accuracy =  0.9223378124548425
Average Accuracy =  0.9223378124548477
```

0.3 3.MultinomialNB

[11]:
```
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time

start_time = time.time()

lp_classifier = LabelPowerset(MultinomialNB())
lp_classifier.fit(x_train, y_train)
lp_predictions = lp_classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

print("Accuracy = ", accuracy_score(y_test, lp_predictions))
print("F1 score = ", f1_score(y_test, lp_predictions, average = "micro"))

lp_predictions = lp_predictions.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, lp_predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],lp_predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
--- 16.43 seconds ---
Accuracy =  0.6013581852333478
F1 score =  0.7074430767268322
Fraction Accuracy =  0.8895390839474059
Average Accuracy =  0.8895390839474064
```

0.4 4. RandomForestClassifier

[12]:
```
from skmultilearn.problem_transform import LabelPowerset
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time

start_time = time.time()

lp_classifier = LabelPowerset(RandomForestClassifier(n_estimators=200,␣
 ↪bootstrap=True, max_features='sqrt',
                                    oob_score=True, random_state=0))
lp_classifier.fit(x_train, y_train)
lp_predictions = lp_classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

print("Accuracy = ", accuracy_score(y_test, lp_predictions))
print("F1 score = ", f1_score(y_test, lp_predictions, average = "micro"))

lp_predictions = lp_predictions.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, lp_predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],lp_predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
--- 851.48 seconds ---
Accuracy =  0.6295333044357752
F1 score =  0.7471199950183698
Fraction Accuracy =  0.9022058469392642
Average Accuracy =  0.902205846939267
```

0.5 5.KNN

[13]:
```
from skmultilearn.problem_transform import LabelPowerset
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time

start_time = time.time()

lp_classifier = LabelPowerset(KNeighborsClassifier(n_neighbors=9,␣
 ↪metric='euclidean'))
lp_classifier.fit(x_train, y_train)
lp_predictions = lp_classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
```

```
print("Accuracy = ", accuracy_score(y_test, lp_predictions))
print("F1 score = ", f1_score(y_test, lp_predictions, average = "micro"))

lp_predictions = lp_predictions.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, lp_predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],lp_predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
--- 47152.8 seconds ---
Accuracy =  0.630544718971247
F1 score =  0.7593813691403909
Fraction Accuracy =  0.9033376679670547
Average Accuracy =  0.9033376679670567
```

[ ]:

Project_ClassifierChain

December 15, 2020

[1]:
```
#import data set
import pandas as pd
import numpy as np
import re
import string

test = pd.read_csv('data/NLP_on_Research_Articles/archive/test.csv')
train = pd.read_csv('data/NLP_on_Research_Articles/archive/train.csv')

print(train.shape)
print(test.shape)
```

```
(20972, 9)
(8989, 3)
```

[2]:
```
#deal with the column names
train.columns = train.columns.str.strip().str.lower().str.replace(' ', '_').str.
 ↪replace('(', '').str.replace(')', '')
print('Train Data shape: ', train.shape)
```

```
Train Data shape:  (20972, 9)
```

[3]:
```
def remove_pattern(text, pattern):
    r = re.findall(pattern, text)
    for i in r:
        text = re.sub(i, "", text)
    return text
```

[4]:
```
#clean data
for column in ['title', 'abstract']:
    train[column] = np.vectorize(remove_pattern)(train[column], "@[\w]*")
    train[column] = np.vectorize(remove_pattern)(train[column], "#[\w]*")
    train[column] = np.vectorize(remove_pattern)(train[column], '[0-9]')
    train[column] = train[column].str.replace("[^a-zA-Z#]", " ")
    train[column] = train[column].apply(lambda x: ' '.join([i for i in x.
 ↪split() if len(i) > 3]))

train['description'] = train['title'] + " " + train['abstract']
```

```
train['description'] = train['description'].str.lower()
```

[5]:
```
#TfidfVectorizer the documents
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words = 'english')
vectors_train = vectorizer.fit_transform(train["description"])
print(vectors_train.shape)
label_train = np.asarray(train[train.columns[3:9]].values)
print(label_train.shape)
```

```
(20972, 46333)
(20972, 6)
```

[6]:
```
#split train data to 2:1
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(vectors_train, label_train,␣
↪test_size=0.33, random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(14051, 46333)
(6921, 46333)
(14051, 6)
(6921, 6)
```

[7]:
```
#Calculate test accuracy by fractional values
def accuracy_fraction(y_test, predicted):
    test_size = y_test.shape[0]
    frac_values = []
    for i in range(0,test_size):
        single_frac = accuracy_score(y_test[i,], predicted[i,])
        frac_values.append(single_frac)
    test_scores = sum(frac_values)/test_size
    return test_scores
```

0.1 1.LR

[7]:
```
# using classifier chains
from skmultilearn.problem_transform import ClassifierChain
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time

start_time = time.time()
```

```
# initialize classifier chains multi-label classifier
classifier = ClassifierChain(LogisticRegression(C=4, solver='newton-cg',␣
 ↪max_iter=4000))
# Training logistic regression model on train data
classifier.fit(x_train, y_train)
# predict
predictions = classifier.predict(x_test)

print("--- %s seconds ---" % round(time.time() - start_time,2))

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
print("f1 score: ", f1_score(y_test, predictions, average='micro'))

predictions = predictions.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
--- 9712.55 seconds ---
Accuracy =  0.6773587631845109
f1 score:  0.8070175438596491
```

```
      ␣
 ↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
 ↪last)

       <ipython-input-7-75a1f8b735c9> in <module>
        21
        22 predictions = predictions.toarray()
   ---> 23 print("Fraction Accuracy = ", accuracy_fraction(y_test, predictions))
        24 accuracy_label = []
        25 for i in list(range(0,6)):


       NameError: name 'accuracy_fraction' is not defined
```

[9]:
```
print("Fraction Accuracy = ", accuracy_fraction(y_test, predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predictions[:,i]))
```

```
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Fraction Accuracy =  0.9218561864855693
Average Accuracy =  0.9218561864855754
```

0.2 2. SVM linear

[10]:
```
# using classifier chains
from skmultilearn.problem_transform import ClassifierChain
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time

start_time = time.time()
# initialize classifier chains multi-label classifier
classifier = ClassifierChain(LinearSVC(random_state=0, C=0.25, penalty='l2'))
# Training logistic regression model on train data
classifier.fit(x_train, y_train)
# predict
predictions = classifier.predict(x_test)

print("--- %s seconds ---" % round(time.time() - start_time,2))

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
print("f1 score: ", f1_score(y_test, predictions, average='micro'))
```

```
--- 391.91 seconds ---
Accuracy =  0.6803930067909262
f1 score:  0.8090449271050283
```

[11]:
```
predictions = predictions.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Fraction Accuracy =  0.9227231132302595
Average Accuracy =  0.9227231132302652
```

[ ]:
```
```

[26]:
```
# using classifier chains
from skmultilearn.problem_transform import ClassifierChain
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import f1_score
import time

start_time = time.time()
# initialize classifier chains multi-label classifier
classifier = ClassifierChain(LinearSVC(random_state=0, C=0.25, penalty='l2'),␣
 ↪order = [0, 1, 2, 3, 4, 5])
# Training logistic regression model on train data
classifier.fit(x_train, y_train)
# predict
predictions = classifier.predict(x_test)

print("--- %s seconds ---" % round(time.time() - start_time,2))

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
print("f1 score: ", f1_score(y_test, predictions, average='micro'))

predictions = predictions.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
--- 312.21 seconds ---
Accuracy =  0.454992053171507
f1 score:  0.5689603812928211
Fraction Accuracy =  0.8257718056157649
Average Accuracy =  0.8257718056157589
```

0.3 3.MultinomialNB

[12]:
```
# using classifier chains
from skmultilearn.problem_transform import ClassifierChain
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time

start_time = time.time()
# initialize classifier chains multi-label classifier
classifier = ClassifierChain(MultinomialNB())
# Training logistic regression model on train data
classifier.fit(x_train, y_train)
# predict
predictions = classifier.predict(x_test)
```

```
print("--- %s seconds ---" % round(time.time() - start_time,2))

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
print("f1 score: ", f1_score(y_test, predictions, average='micro'))

predictions = predictions.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
--- 163.1 seconds ---
Accuracy =  0.5662476520733998
f1 score:  0.7166963918232954
Fraction Accuracy =  0.9005442373452719
Average Accuracy =  0.9005442373452777
```

0.4 4. RandomForest

[13]:
```
# using classifier chains
from skmultilearn.problem_transform import ClassifierChain
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time

start_time = time.time()
# initialize classifier chains multi-label classifier
classifier = ClassifierChain(RandomForestClassifier(n_estimators=200,␣
 ↪bootstrap=True, max_features='sqrt',
                                    oob_score=True, random_state=0))
# Training logistic regression model on train data
classifier.fit(x_train, y_train)
# predict
predictions = classifier.predict(x_test)

print("--- %s seconds ---" % round(time.time() - start_time,2))

# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
print("f1 score: ", f1_score(y_test, predictions, average='micro'))

predictions = predictions.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predictions))
```

```
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
--- 20828.24 seconds ---
Accuracy =  0.6250541829215431
f1 score:  0.7764130843464093
Fraction Accuracy =  0.9140779270818189
Average Accuracy =  0.9140779270818283
```

[ ]:

[ ]:

[ ]:

[ ]:

0.5 Adapted Algorithm

#https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff

[7]:
```
from skmultilearn.adapt import MLkNN
from sklearn.metrics import accuracy_score
import time

start_time = time.time()
clf = MLkNN(k=10)
# train
clf.fit(x_train, y_train)
# predict
predictions = clf.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",accuracy_score(y_test,predictions))
```

```
Accuracy =  0.6143620864036989
```

[18]:
```
from skmultilearn.adapt import MLkNN
from sklearn.metrics import accuracy_score
import time

n = list(range(1,15))
test_scores = []
frac_score = []
for i in n:
```

```
    start_time = time.time()
    model = MLkNN(k=i)
    model.fit(x_train,y_train)
    predicted= model.predict(x_test)
    knn_scores = accuracy_score(y_test, predicted)
    print("--- %s seconds ---" % round(time.time() - start_time,2))
    # accuracy
    print("Accuracy = ",knn_scores)
    predicted = predicted.toarray()
    print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))

    test_scores.append(knn_scores)
    frac_score.append(accuracy_fraction(y_test, predicted))
print(test_scores)
print(frac_score)
```

```
--- 25.12 seconds ---
Accuracy =  0.5435630689206762
Fraction Accuracy =  0.8773057843278884
--- 23.98 seconds ---
Accuracy =  0.5217454125126427
Fraction Accuracy =  0.8761498820016345
--- 27.41 seconds ---
Accuracy =  0.5704377980060685
Fraction Accuracy =  0.8922361893753252
--- 29.55 seconds ---
Accuracy =  0.5700043346337235
Fraction Accuracy =  0.8918990511968351
--- 29.55 seconds ---
Accuracy =  0.5896546741800318
Fraction Accuracy =  0.8992920098251626
--- 30.82 seconds ---
Accuracy =  0.5915330154601937
Fraction Accuracy =  0.8993883350190184
--- 29.42 seconds ---
Accuracy =  0.6016471608149111
Fraction Accuracy =  0.902952367191632
--- 28.35 seconds ---
Accuracy =  0.6062707701199249
Fraction Accuracy =  0.9043490825025228
--- 28.18 seconds ---
Accuracy =  0.6119057939604103
Fraction Accuracy =  0.9057939604103364
--- 27.44 seconds ---
Accuracy =  0.6143620864036989
Fraction Accuracy =  0.906323748976536
--- 27.63 seconds ---
```

```
Accuracy =  0.6179742811732408
Fraction Accuracy =  0.9077686268843547
--- 28.41 seconds ---
Accuracy =  0.6185522323363676
Fraction Accuracy =  0.9079612772720635
--- 28.68 seconds ---
Accuracy =  0.6217309637335645
Fraction Accuracy =  0.9096710494629804
--- 28.79 seconds ---
Accuracy =  0.6186967201271493
Fraction Accuracy =  0.9093098299860239
[0.5435630689206762, 0.5217454125126427, 0.5704377980060685, 0.5700043346337235,
0.5896546741800318, 0.5915330154601937, 0.6016471608149111, 0.6062707701199249,
0.6119057939604103, 0.6143620864036989, 0.6179742811732408, 0.6185522323363676,
0.6217309637335645, 0.6186967201271493]
[0.8773057843278884, 0.8761498820016345, 0.8922361893753252, 0.8918990511968351,
0.8992920098251626, 0.8993883350190184, 0.902952367191632, 0.9043490825025228,
0.9057939604103364, 0.906323748976536, 0.9077686268843547, 0.9079612772720635,
0.9096710494629804, 0.9093098299860239]
```

[17]:
```python
from skmultilearn.adapt import BRkNNaClassifier
from sklearn.metrics import accuracy_score
import time

n = list(range(1,16))
test_scores = []
frac_score = []
for i in n:
    start_time = time.time()
    model = BRkNNaClassifier(k=i)
    model.fit(x_train,y_train)
    predicted= model.predict(x_test)
    knn_scores = accuracy_score(y_test, predicted)
    print("--- %s seconds ---" % round(time.time() - start_time,2))
    # accuracy
    print("Accuracy = ",knn_scores)
    predicted = predicted.toarray()
    print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))

    test_scores.append(knn_scores)
    frac_score.append(accuracy_fraction(y_test, predicted))
print(test_scores)
print(frac_score)
```

```
--- 8.14 seconds ---
Accuracy =  0.5435630689206762
Fraction Accuracy =  0.8773057843278884
--- 6.08 seconds ---
```

```
Accuracy =  0.49125848865770844
Fraction Accuracy =  0.8818571497375054
--- 6.23 seconds ---
Accuracy =  0.5704377980060685
Fraction Accuracy =  0.8922361893753252
--- 6.77 seconds ---
Accuracy =  0.5465973125270914
Fraction Accuracy =  0.8942830997447297
--- 7.02 seconds ---
Accuracy =  0.5879208206906517
Fraction Accuracy =  0.8998458796898258
--- 6.81 seconds ---
Accuracy =  0.5668256032365265
Fraction Accuracy =  0.8995569041082617
--- 7.53 seconds ---
Accuracy =  0.5971680393006791
Fraction Accuracy =  0.9032895053701229
--- 7.8 seconds ---
Accuracy =  0.5828637480132929
Fraction Accuracy =  0.9033617492655127
--- 7.22 seconds ---
Accuracy =  0.603958965467418
Fraction Accuracy =  0.905336415739529
--- 7.15 seconds ---
Accuracy =  0.5853200404565814
Fraction Accuracy =  0.9041082695178837
--- 6.99 seconds ---
Accuracy =  0.6043924288397631
Fraction Accuracy =  0.9068776188412
--- 7.15 seconds ---
Accuracy =  0.5950007224389539
Fraction Accuracy =  0.906757212348881
--- 7.82 seconds ---
Accuracy =  0.6071376968646149
Fraction Accuracy =  0.907961277272064
--- 7.93 seconds ---
Accuracy =  0.5942782834850455
Fraction Accuracy =  0.9067090497519545
--- 7.44 seconds ---
Accuracy =  0.60757116023696
Fraction Accuracy =  0.9083706593459442
[0.5435630689206762, 0.49125848865770844, 0.5704377980060685,
0.5465973125270914, 0.5879208206906517, 0.5668256032365265, 0.5971680393006791,
0.5828637480132929, 0.603958965467418, 0.5853200404565814, 0.6043924288397631,
0.5950007224389539, 0.6071376968646149, 0.5942782834850455, 0.60757116023696]
[0.8773057843278884, 0.8818571497375054, 0.8922361893753252, 0.8942830997447297,
0.8998458796898258, 0.8995569041082617, 0.9032895053701229, 0.9033617492655127,
0.905336415739529, 0.9041082695178837, 0.9068776188412, 0.906757212348881,
```

```
0.907961277272064, 0.9067090497519545, 0.9083706593459442]
```

[19]:
```
from skmultilearn.adapt import BRkNNbClassifier
from sklearn.metrics import accuracy_score
import time

n = list(range(1,16))
test_scores = []
frac_score = []
for i in n:
    start_time = time.time()
    model = BRkNNbClassifier(k=i)
    model.fit(x_train,y_train)
    predicted= model.predict(x_test)
    knn_scores = accuracy_score(y_test, predicted)
    print("--- %s seconds ---" % round(time.time() - start_time,2))
    # accuracy
    print("Accuracy = ",knn_scores)
    predicted = predicted.toarray()
    print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))

    test_scores.append(knn_scores)
    frac_score.append(accuracy_fraction(y_test, predicted))
print(test_scores)
print(frac_score)
```

```
--- 9.3 seconds ---
Accuracy =  0.007224389539083947
Fraction Accuracy =  0.5923276983094603
--- 8.53 seconds ---
Accuracy =  0.006068487212830516
Fraction Accuracy =  0.5802148051822718
--- 8.68 seconds ---
Accuracy =  0.009825169773154169
Fraction Accuracy =  0.6115686557818911
--- 9.26 seconds ---
Accuracy =  0.01242595000722439
Fraction Accuracy =  0.6088474690555062
--- 9.98 seconds ---
Accuracy =  0.021095217454125126
Fraction Accuracy =  0.6224774839859105
--- 9.4 seconds ---
Accuracy =  0.025429851177575496
Fraction Accuracy =  0.6227664595674761
--- 9.02 seconds ---
Accuracy =  0.03482155757838463
Fraction Accuracy =  0.6330491740114403
--- 9.59 seconds ---
```

```
Accuracy =  0.03800028897558156
Fraction Accuracy =  0.6345662958146503
--- 9.13 seconds ---
Accuracy =  0.04695853200404566
Fraction Accuracy =  0.6415498723691001
--- 9.47 seconds ---
Accuracy =  0.051871116890622744
Fraction Accuracy =  0.6436449453354364
--- 9.65 seconds ---
Accuracy =  0.05981794538361508
Fraction Accuracy =  0.649015074892823
--- 9.77 seconds ---
Accuracy =  0.06545296922410056
Fraction Accuracy =  0.6509656600683774
--- 9.87 seconds ---
Accuracy =  0.07282184655396619
Fraction Accuracy =  0.6558782449549548
--- 9.87 seconds ---
Accuracy =  0.07990174830226845
Fraction Accuracy =  0.6584549438905628
--- 9.97 seconds ---
Accuracy =  0.08625921109666233
Fraction Accuracy =  0.6628618215094048
[0.007224389539083947, 0.006068487212830516, 0.009825169773154169,
0.01242595000722439, 0.021095217454125126, 0.025429851177575496,
0.03482155757838463, 0.03800028897558156, 0.04695853200404566,
0.051871116890622744, 0.05981794538361508, 0.06545296922410056,
0.07282184655396619, 0.07990174830226845, 0.08625921109666233]
[0.5923276983094603, 0.5802148051822718, 0.6115686557818911, 0.6088474690555062,
0.6224774839859105, 0.6227664595674761, 0.6330491740114403, 0.6345662958146503,
0.6415498723691001, 0.6436449453354364, 0.649015074892823, 0.6509656600683774,
0.6558782449549548, 0.6584549438905628, 0.6628618215094048]
```

[ ]:

[ ]:

[ ]:

[1]:
```python
import pandas as pd

x_train_pca = pd.read_csv('data/NLP_on_Research_Articles/archive/pca_train.csv')

print(x_train_pca.shape)
```

```
(20972, 10301)
```

[4]:
```
import numpy as np
train = pd.read_csv('data/NLP_on_Research_Articles/archive/train.csv')
label_train = np.asarray(train[train.columns[3:9]].values)
print(label_train.shape)
```

```
(20972, 6)
```

[13]:
```
#PCA split
#split train data to 2:1
from sklearn.model_selection import train_test_split
x_train_pca = np.matrix(x_train_pca)
pca_x_train, pca_x_test, pca_y_train, pca_y_test =␣
 ↪train_test_split(x_train_pca, label_train, test_size=0.33, random_state=42)
print(pca_x_train.shape)
print(pca_x_test.shape)
print(pca_y_train.shape)
print(pca_y_test.shape)
```

```
(14051, 10301)
(6921, 10301)
(14051, 6)
(6921, 6)
```

[14]:
```
type(pca_x_train)
```

[14]:
```
numpy.matrix
```

[1]:
```
from skmultilearn.adapt import MLTSVM
from sklearn.metrics import accuracy_score
import time

start_time = time.time()
model = MLTSVM(c_k = 2**-1)
model.fit(pca_x_train,pca_y_train)
predicted= model.predict(pca_x_test)
knn_scores = accuracy_score(pca_y_test, predicted)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",knn_scores)
#predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(pca_y_test, predicted))
```

[9]:
```
from skmultilearn.adapt import MLARAM
from sklearn.metrics import accuracy_score
import time

start_time = time.time()
```

```
model = MLARAM(threshold=0.05, vigilance=0.9)
model.fit(x_train, y_train)
predicted= model.predict(x_test)
test_scores = accuracy_score(y_test, predicted)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",test_scores)
#predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 187.63 seconds ---
Accuracy =  0.019361363964744978
Fraction Accuracy =  0.6900736887733022
```

[11]:
```
from skmultilearn.adapt import MLARAM
from sklearn.metrics import accuracy_score
import time

start_time = time.time()
model = MLARAM(threshold=0.01, vigilance=0.9)
model.fit(x_train, y_train)
predicted= model.predict(x_test)
test_scores = accuracy_score(y_test, predicted)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",test_scores)
#predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 183.99 seconds ---
Accuracy =  0.22973558734286953
Fraction Accuracy =  0.7583923325145748
```

[14]:
```
from skmultilearn.adapt import MLARAM
from sklearn.metrics import accuracy_score
import time

start_time = time.time()
model = MLARAM(threshold=0.005, vigilance=0.8)
model.fit(x_train, y_train)
predicted= model.predict(x_test)
test_scores = accuracy_score(y_test, predicted)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",test_scores)
#predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 207.37 seconds ---
Accuracy =  0.22973558734286953
Fraction Accuracy =  0.7583923325145748
```

[10]:
```python
from sklearn.naive_bayes import MultinomialNB
from skmultilearn.ensemble import RakelD
from sklearn.metrics import accuracy_score

import time

start_time = time.time()
classifier = RakelD(
    base_classifier=MultinomialNB(),
    base_classifier_require_dense=[True, True],
    labelset_size=4
)

classifier.fit(x_train, y_train)
predicted = classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",accuracy_score(y_test, predicted))
predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 37.61 seconds ---
Accuracy =  0.5785291142898425
Fraction Accuracy =  0.8954630833694512
```

[12]:
```python
from sklearn.naive_bayes import MultinomialNB
from skmultilearn.ensemble import RakelD
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

import time

start_time = time.time()
classifier = RakelD(
    base_classifier=LinearSVC(random_state=0, C=0.25, penalty='l2'),
    base_classifier_require_dense=[True, True],
    labelset_size=6
)

classifier.fit(x_train, y_train)
predicted = classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
```

```
print("Accuracy = ",accuracy_score(y_test, predicted))
predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 31.02 seconds ---
Accuracy =  0.6828492992342147
Fraction Accuracy =  0.9223378124548425
```

[22]:
```
from sklearn.naive_bayes import MultinomialNB
from skmultilearn.ensemble import RakelD
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

import time

start_time = time.time()
classifier = RakelD(
    base_classifier=LinearSVC(random_state=0, C=0.25, penalty='l2'),
    base_classifier_require_dense=[True, True],
    labelset_size=3
)

classifier.fit(x_train, y_train)
predicted = classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",accuracy_score(y_test, predicted))
predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 80.29 seconds ---
Accuracy =  0.6736020806241872
Fraction Accuracy =  0.9240235033472929
```

[8]:
```
from sklearn.naive_bayes import MultinomialNB
from skmultilearn.ensemble import RakelD
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

import time

start_time = time.time()
classifier = RakelD(
    base_classifier=LinearSVC(random_state=0, C=0.25, penalty='l2'),
    base_classifier_require_dense=[True, True],
    labelset_size=2
)
```

```
classifier.fit(x_train, y_train)
predicted = classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",accuracy_score(y_test, predicted))
predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 56.42 seconds ---
Accuracy =  0.6603092038722728
Fraction Accuracy =  0.9249145113904441
```

[15]:
```
from sklearn.naive_bayes import MultinomialNB
from skmultilearn.ensemble import RakelO
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

import time

start_time = time.time()
classifier = RakelO(
    base_classifier=LinearSVC(random_state=0, C=0.25, penalty='l2'),
    base_classifier_require_dense=[True, True],
    labelset_size=6,
    model_count=12
)

classifier.fit(x_train, y_train)
predicted = classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",accuracy_score(y_test, predicted))
predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 290.4 seconds ---
Accuracy =  0.6828492992342147
Fraction Accuracy =  0.9223378124548425
```

[23]:
```
from sklearn.naive_bayes import MultinomialNB
from skmultilearn.ensemble import RakelO
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

import time
```

```
start_time = time.time()
classifier = RakelO(
    base_classifier=LinearSVC(random_state=0, C=0.25, penalty='l2'),
    base_classifier_require_dense=[True, True],
    labelset_size=3,
    model_count=12
)

classifier.fit(x_train, y_train)
predicted = classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",accuracy_score(y_test, predicted))
predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 477.13 seconds ---
Accuracy =  0.6785146655107643
Fraction Accuracy =  0.9254442999566467
```

[9]:
```
from sklearn.naive_bayes import MultinomialNB
from skmultilearn.ensemble import RakelO
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

import time

start_time = time.time()
classifier = RakelO(
    base_classifier=LinearSVC(random_state=0, C=0.25, penalty='l2'),
    base_classifier_require_dense=[True, True],
    labelset_size=2,
    model_count=12
)

classifier.fit(x_train, y_train)
predicted = classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",accuracy_score(y_test, predicted))
predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 198.64 seconds ---
Accuracy =  0.6559745701488224
Fraction Accuracy =  0.9226749506333292
```

[10]:
```
from sklearn.naive_bayes import MultinomialNB
from skmultilearn.ensemble import RakelO
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score

import time

start_time = time.time()
classifier = RakelO(
    base_classifier=LinearSVC(random_state=0, C=0.25, penalty='l2'),
    base_classifier_require_dense=[True, True],
    labelset_size=4,
    model_count=12
)

classifier.fit(x_train, y_train)
predicted = classifier.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
# accuracy
print("Accuracy = ",accuracy_score(y_test, predicted))
predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
```

```
--- 216.24 seconds ---
Accuracy =  0.6792371044646728
Fraction Accuracy =  0.9233492269903139
```

[ ]:
```

```

Project_MLP

December 15, 2020

[1]:
```
#import data set
import pandas as pd
import numpy as np
import re
import string

test = pd.read_csv('data/NLP_on_Research_Articles/archive/test.csv')
train = pd.read_csv('data/NLP_on_Research_Articles/archive/train.csv')

print(train.shape)
print(test.shape)
```

```
(20972, 9)
(8989, 3)
```

[2]:
```
#deal with the column names
train.columns = train.columns.str.strip().str.lower().str.replace(' ', '_').str.
 ↪replace('(', '').str.replace(')', '')
print('Train Data shape: ', train.shape)
```

```
Train Data shape:  (20972, 9)
```

[3]:
```
def remove_pattern(text, pattern):
    r = re.findall(pattern, text)
    for i in r:
        text = re.sub(i, "", text)
    return text
```

[4]:
```
#clean data
for column in ['title', 'abstract']:
    train[column] = np.vectorize(remove_pattern)(train[column], "@[\w]*")
    train[column] = np.vectorize(remove_pattern)(train[column], "#[\w]*")
    train[column] = np.vectorize(remove_pattern)(train[column], '[0-9]')
    train[column] = train[column].str.replace("[^a-zA-Z#]", " ")
    train[column] = train[column].apply(lambda x: ' '.join([i for i in x.
 ↪split() if len(i) > 3]))

train['description'] = train['title'] + " " + train['abstract']
```

```
train['description'] = train['description'].str.lower()
```

[5]:
```
#TfidfVectorizer the documents
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words = 'english')
vectors_train = vectorizer.fit_transform(train["description"])
print(vectors_train.shape)
label_train = np.asarray(train[train.columns[3:9]].values)
print(label_train.shape)
```

```
(20972, 46333)
(20972, 6)
```

[6]:
```
#split train data to 2:1
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(vectors_train, label_train,␣
↪test_size=0.33, random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(14051, 46333)
(6921, 46333)
(14051, 6)
(6921, 6)
```

[7]:
```
#Calculate test accuracy by fractional values
def accuracy_fraction(y_test, predicted):
    test_size = y_test.shape[0]
    frac_values = []
    for i in range(0,test_size):
        single_frac = accuracy_score(y_test[i,], predicted[i,])
        frac_values.append(single_frac)
    test_scores = sum(frac_values)/test_size
    return test_scores
```

0.1 Binary relevance

[ ]:
```
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import time

start_time = time.time()
```

```
#mlp = BinaryRelevance(MLPClassifier(hidden_layer_sizes=(100,), max_iter=100,␣
↪alpha=1e-4, solver='sgd',
                      #batch_size=300, learning_rate='constant',␣
↪learning_rate_init = 0.001,
                      #verbose=False, random_state=1, activation='logistic'))
mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=100, alpha=1e-4,␣
↪solver='sgd',
                      batch_size=300, learning_rate='constant',␣
↪learning_rate_init = 0.001,
                      verbose=False, random_state=1, activation='logistic')
print("MLP trained")
mlp.fit(x_train, y_train[:,0])
predicted = mlp.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Accuracy = ", accuracy_score(y_test[:,0], predicted))

print("Fraction Accuracy = ", accuracy_fraction(y_test[:,0], predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

[10]:
```
predicted
```

[10]:
```
<6921x6 sparse matrix of type '<class 'numpy.int64'>'
        with 0 stored elements in Compressed Sparse Column format>
```

[ ]:
```

```

0.2 label powerset

[14]:
```
from skmultilearn.problem_transform import LabelPowerset
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time

start_time = time.time()

lp_classifier = LabelPowerset(MLPClassifier(hidden_layer_sizes=(100,),␣
↪max_iter=100, alpha=1e-4, solver='sgd',
                      batch_size=300, learning_rate='constant',␣
↪learning_rate_init = 0.001,
                      verbose=False, random_state=1, activation='logistic'))
lp_classifier.fit(x_train, y_train)
lp_predictions = lp_classifier.predict(x_test)
```

```
print("--- %s seconds ---" % round(time.time() - start_time,2))

print("Accuracy = ", accuracy_score(y_test, lp_predictions))
print("F1 score = ", f1_score(y_test, lp_predictions, average = "micro"))

lp_predictions = lp_predictions.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, lp_predictions))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],lp_predictions[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
--- 1771.5 seconds ---
Accuracy =  0.2505418292154313
F1 score =  0.26127567014266523
Fraction Accuracy =  0.72193324664066
Average Accuracy =  0.7219332466406589
```

0.3 ClassifierChain

[52]:
```
from skmultilearn.problem_transform import ClassifierChain
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import time

start_time = time.time()

mlp = MLPClassifier(hidden_layer_sizes=(300,), max_iter=100, alpha=2,␣
↪solver='sgd',
                    batch_size=300, learning_rate='constant',␣
↪learning_rate_init = 0.001,
                    verbose=False, random_state=1, activation='logistic')
mlp.fit(x_train, y_train)
predicted = mlp.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

print("Accuracy = ", accuracy_score(y_test, predicted))

#predicted = predicted.toarray()
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
/opt/anaconda3/lib/python3.7/site-
```

```
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)

--- 1278.81 seconds ---
Accuracy =  0.0
Fraction Accuracy =  0.7902518903819147
Average Accuracy =  0.7902518903819294
```

0.4 keras

[8]:
```
import numpy as np
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

[10]:
```
# get the model
def get_model(n_inputs, n_outputs):
    model = Sequential()
    model.add(Dense(300, input_dim=n_inputs, kernel_initializer='normal',␣
 ↪activation='relu'))
    #model.add(Dense(100, activation='relu'))
    model.add(Dense(n_outputs, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam')
    print(model.summary())
    return model
```

[82]:
```
# get the model
def get_rnn_model(n_inputs, n_outputs):
    model = Sequential()
    model.add(Dense(300, input_dim=n_inputs, kernel_initializer='normal',␣
 ↪activation='relu'))
    #model.add(LSTM(128, return_sequences=True, input_shape=(100,n_inputs)))
    model.add(LSTM(128))
    model.add(Dense(n_outputs, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam')
    print(model.summary())
    return model
```

[11]:
```
n_inputs, n_outputs = x_train.shape[1], y_train.shape[1]
print(n_inputs)
print(n_outputs)
print(x_train.shape[0])
x_train = x_train.toarray()
x_test = x_test.toarray()
```

```
46333
6
14051
```

[15]:
```
import time

# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=10, batch_size=300)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")


from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 300)               13900200
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 1806
=================================================================
Total params: 13,902,006
Trainable params: 13,902,006
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
47/47 [==============================] - 28s 604ms/step - loss: 0.5084
Epoch 2/10
47/47 [==============================] - 21s 445ms/step - loss: 0.2845
Epoch 3/10
47/47 [==============================] - 20s 433ms/step - loss: 0.1958
```

```
Epoch 4/10
47/47 [==============================] - 20s 429ms/step - loss: 0.1531
Epoch 5/10
47/47 [==============================] - 20s 429ms/step - loss: 0.1239
Epoch 6/10
47/47 [==============================] - 21s 451ms/step - loss: 0.1004
Epoch 7/10
47/47 [==============================] - 20s 426ms/step - loss: 0.0807
Epoch 8/10
47/47 [==============================] - 20s 434ms/step - loss: 0.0648
Epoch 9/10
47/47 [==============================] - 20s 435ms/step - loss: 0.0518
Epoch 10/10
47/47 [==============================] - 20s 433ms/step - loss: 0.0413
--- 265.7 seconds ---
Done
```

[16]:
```
from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Accuracy =  0.658864325964456
Fraction Accuracy =  0.923252901796454
Average Accuracy =  0.9232529017964649
```

[17]:
```
import time

# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=20, batch_size=300)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")


from sklearn.metrics import accuracy_score
```

```
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 300)               13900200
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 1806
=================================================================
Total params: 13,902,006
Trainable params: 13,902,006
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/20
47/47 [==============================] - 28s 593ms/step - loss: 0.5157
Epoch 2/20
47/47 [==============================] - 24s 510ms/step - loss: 0.2833
Epoch 3/20
47/47 [==============================] - 22s 465ms/step - loss: 0.1946
Epoch 4/20
47/47 [==============================] - 21s 442ms/step - loss: 0.1517
Epoch 5/20
47/47 [==============================] - 22s 462ms/step - loss: 0.1222
Epoch 6/20
47/47 [==============================] - 21s 440ms/step - loss: 0.0985
Epoch 7/20
47/47 [==============================] - 20s 432ms/step - loss: 0.0788
Epoch 8/20
47/47 [==============================] - 20s 428ms/step - loss: 0.0629
Epoch 9/20
47/47 [==============================] - 20s 435ms/step - loss: 0.0501
Epoch 10/20
47/47 [==============================] - 20s 431ms/step - loss: 0.0400
Epoch 11/20
47/47 [==============================] - 20s 417ms/step - loss: 0.0321
Epoch 12/20
47/47 [==============================] - 20s 426ms/step - loss: 0.0259
```

```
Epoch 13/20
47/47 [==============================] - 20s 420ms/step - loss: 0.0211
Epoch 14/20
47/47 [==============================] - 20s 434ms/step - loss: 0.0173
Epoch 15/20
47/47 [==============================] - 20s 417ms/step - loss: 0.0143
Epoch 16/20
47/47 [==============================] - 20s 421ms/step - loss: 0.0120
Epoch 17/20
47/47 [==============================] - 20s 420ms/step - loss: 0.0101
Epoch 18/20
47/47 [==============================] - 20s 420ms/step - loss: 0.0086
Epoch 19/20
47/47 [==============================] - 20s 425ms/step - loss: 0.0074
Epoch 20/20
47/47 [==============================] - 21s 444ms/step - loss: 0.0064
--- 471.46 seconds ---
Done
Accuracy =  0.6448490102586332
Fraction Accuracy =  0.9201704955931125
Average Accuracy =  0.9201704955931224
```

[18]:
```python
import time

# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=20, batch_size=300)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")


from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential_3"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 300)               13900200
_________________________________________________________________
dense_7 (Dense)              (None, 6)                 1806
=================================================================
Total params: 13,902,006
Trainable params: 13,902,006
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/20
110/110 [==============================] - 30s 274ms/step - loss: 0.3999
Epoch 2/20
110/110 [==============================] - 24s 222ms/step - loss: 0.1891
Epoch 3/20
110/110 [==============================] - 23s 212ms/step - loss: 0.1328
Epoch 4/20
110/110 [==============================] - 23s 210ms/step - loss: 0.0948
Epoch 5/20
110/110 [==============================] - 24s 220ms/step - loss: 0.0659
Epoch 6/20
110/110 [==============================] - 24s 215ms/step - loss: 0.0451
Epoch 7/20
110/110 [==============================] - 23s 213ms/step - loss: 0.0307
Epoch 8/20
110/110 [==============================] - 23s 213ms/step - loss: 0.0210
Epoch 9/20
110/110 [==============================] - 23s 209ms/step - loss: 0.0147
Epoch 10/20
110/110 [==============================] - 24s 216ms/step - loss: 0.0106
Epoch 11/20
110/110 [==============================] - 23s 213ms/step - loss: 0.0079
Epoch 12/20
110/110 [==============================] - 23s 212ms/step - loss: 0.0060
Epoch 13/20
110/110 [==============================] - 23s 206ms/step - loss: 0.0047
Epoch 14/20
110/110 [==============================] - 23s 208ms/step - loss: 0.0038
Epoch 15/20
110/110 [==============================] - 23s 207ms/step - loss: 0.0031
Epoch 16/20
110/110 [==============================] - 23s 209ms/step - loss: 0.0026
Epoch 17/20
110/110 [==============================] - 23s 209ms/step - loss: 0.0022
Epoch 18/20
110/110 [==============================] - 24s 217ms/step - loss: 0.0018
```

```
Epoch 19/20
110/110 [==============================] - 23s 208ms/step - loss: 0.0016
Epoch 20/20
110/110 [==============================] - 23s 211ms/step - loss: 0.0014
--- 522.2 seconds ---
Done
Accuracy =  0.6376246207195492
Fraction Accuracy =  0.9185811298945155
Average Accuracy =  0.9185811298945238
```

[19]:
```
import time

# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=10, batch_size=300)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")


from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 300)               13900200
_________________________________________________________________
dense_9 (Dense)              (None, 6)                 1806
=================================================================
Total params: 13,902,006
Trainable params: 13,902,006
Non-trainable params: 0
_________________________________________________________________
None
```

```
Epoch 1/10
47/47 [==============================] - 22s 468ms/step - loss: 0.5140
Epoch 2/10
47/47 [==============================] - 18s 387ms/step - loss: 0.2848
Epoch 3/10
47/47 [==============================] - 19s 408ms/step - loss: 0.1948
Epoch 4/10
47/47 [==============================] - 18s 383ms/step - loss: 0.1523
Epoch 5/10
47/47 [==============================] - 18s 375ms/step - loss: 0.1228
Epoch 6/10
47/47 [==============================] - 18s 386ms/step - loss: 0.0987
Epoch 7/10
47/47 [==============================] - 18s 386ms/step - loss: 0.0791
Epoch 8/10
47/47 [==============================] - 19s 410ms/step - loss: 0.0629
Epoch 9/10
47/47 [==============================] - 18s 381ms/step - loss: 0.0501
Epoch 10/10
47/47 [==============================] - 19s 396ms/step - loss: 0.0399
--- 233.3 seconds ---
Done
Accuracy =  0.6616095939893079
Fraction Accuracy =  0.9241439098396075
Average Accuracy =  0.9241439098396187
```

[20]:
```python
import time

# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=10, batch_size=400)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")


from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
```

```
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 300)               13900200
_________________________________________________________________
dense_11 (Dense)             (None, 6)                 1806
=================================================================
Total params: 13,902,006
Trainable params: 13,902,006
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
36/36 [==============================] - 15s 419ms/step - loss: 0.5481
Epoch 2/10
36/36 [==============================] - 11s 303ms/step - loss: 0.3359
Epoch 3/10
36/36 [==============================] - 12s 328ms/step - loss: 0.2290
Epoch 4/10
36/36 [==============================] - 11s 293ms/step - loss: 0.1776
Epoch 5/10
36/36 [==============================] - 11s 311ms/step - loss: 0.1468
Epoch 6/10
36/36 [==============================] - 11s 310ms/step - loss: 0.1234
Epoch 7/10
36/36 [==============================] - 10s 281ms/step - loss: 0.1039
Epoch 8/10
36/36 [==============================] - 10s 269ms/step - loss: 0.0872
Epoch 9/10
36/36 [==============================] - 10s 264ms/step - loss: 0.0729
Epoch 10/10
36/36 [==============================] - 10s 285ms/step - loss: 0.0609
--- 142.87 seconds ---
Done
Accuracy =  0.6642103742233781
Fraction Accuracy =  0.9251071617781534
Average Accuracy =  0.9251071617781631
```

[21]:
```
import time

# define model
model = get_model(n_inputs, n_outputs)
```

```
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=10, batch_size=500)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")


from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 300)               13900200
_________________________________________________________________
dense_13 (Dense)             (None, 6)                 1806
=================================================================
Total params: 13,902,006
Trainable params: 13,902,006
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
29/29 [==============================] - 24s 844ms/step - loss: 0.5816
Epoch 2/10
29/29 [==============================] - 19s 648ms/step - loss: 0.3748
Epoch 3/10
29/29 [==============================] - 19s 649ms/step - loss: 0.2628
Epoch 4/10
29/29 [==============================] - 18s 628ms/step - loss: 0.2015
Epoch 5/10
29/29 [==============================] - 19s 647ms/step - loss: 0.1662
Epoch 6/10
29/29 [==============================] - 18s 637ms/step - loss: 0.1416
Epoch 7/10
```

```
29/29 [==============================] - 19s 649ms/step - loss: 0.1217
Epoch 8/10
29/29 [==============================] - 19s 643ms/step - loss: 0.1047
Epoch 9/10
29/29 [==============================] - 19s 652ms/step - loss: 0.0900
Epoch 10/10
29/29 [==============================] - 18s 626ms/step - loss: 0.0771
--- 246.57 seconds ---
Done
Accuracy =  0.6623320329432163
Fraction Accuracy =  0.9250108365842985
Average Accuracy =  0.9250108365843087
```

0.5 case selected

[12]:
```
# get the model
def get_model(n_inputs, n_outputs):
    model = Sequential()
    model.add(Dense(300, input_dim=n_inputs, kernel_initializer='normal',␣
 ↪activation='relu'))
    model.add(Dropout(0.3))
    #model.add(Dense(100, activation='relu'))
    model.add(Dense(n_outputs, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam')
    print(model.summary())
    return model
```

[13]:
```
import time
from tensorflow.keras.layers import Dropout


# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=10, batch_size=500)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")


from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
```

```
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 300)               13900200
_________________________________________________________________
dropout (Dropout)            (None, 300)               0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 1806
=================================================================
Total params: 13,902,006
Trainable params: 13,902,006
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
29/29 [==============================] - 25s 871ms/step - loss: 0.5861
Epoch 2/10
29/29 [==============================] - 18s 632ms/step - loss: 0.3880
Epoch 3/10
29/29 [==============================] - 18s 617ms/step - loss: 0.2782
Epoch 4/10
29/29 [==============================] - 17s 596ms/step - loss: 0.2168
Epoch 5/10
29/29 [==============================] - 18s 608ms/step - loss: 0.1807
Epoch 6/10
29/29 [==============================] - 17s 594ms/step - loss: 0.1559
Epoch 7/10
29/29 [==============================] - 18s 606ms/step - loss: 0.1364
Epoch 8/10
29/29 [==============================] - 18s 621ms/step - loss: 0.1206
Epoch 9/10
29/29 [==============================] - 18s 614ms/step - loss: 0.1059
Epoch 10/10
29/29 [==============================] - 17s 602ms/step - loss: 0.0931
--- 229.76 seconds ---
Done
Accuracy =  0.6672446178297934
Fraction Accuracy =  0.9258296007320624
Average Accuracy =  0.9258296007320714
```

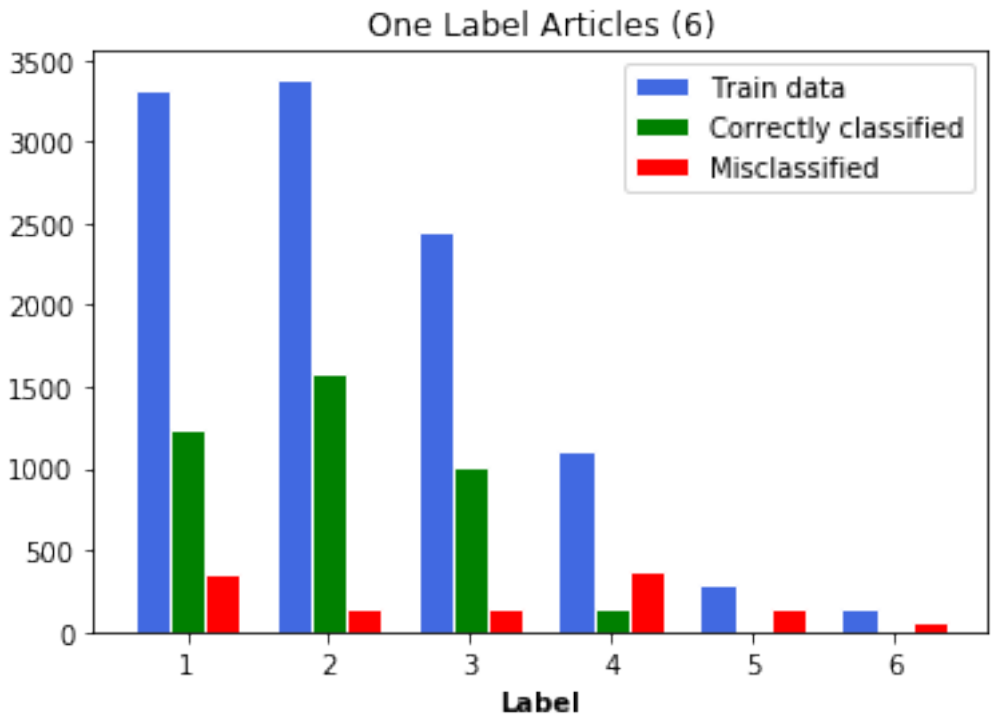
[14]:
```
misclassification = (y_test == predicted).all(axis=1)
i, = np.where(misclassification == False)
j, = np.where(misclassification == True)
mis_y_test = y_test[i]
correct_y_test = y_test[j]
print(len(mis_y_test))
print(len(correct_y_test))
print(len(y_test))
```

```
2303
4618
6921
```

[17]:
```
#mis articles that only belong to 1 label.
x = np.array([[1, 0, 0, 0, 0, 0],
              [0, 1, 0, 0, 0, 0],
              [0, 0, 1, 0, 0, 0],
              [0, 0, 0, 1, 0, 0],
              [0, 0, 0, 0, 1, 0],
              [0, 0, 0, 0, 0, 1]])
c = x.shape[0]
mis_num = []
correct_num = []
train_num = []

for i in range(0,c):
    each_combine = np.repeat(([x[i,:]]), len(mis_y_test), axis=0)
    paired_mis_num = accuracy_score(mis_y_test, each_combine, False)
    mis_num.append(paired_mis_num)

    each_combine = np.repeat(([x[i,:]]), len(correct_y_test), axis=0)
    paired_correct_num = accuracy_score(correct_y_test, each_combine, False)
    correct_num.append(paired_correct_num)

    each_combine = np.repeat(([x[i,:]]), len(y_train), axis=0)
    paired_train_num = accuracy_score(y_train, each_combine, False)
    train_num.append(paired_train_num)

print(mis_num)
print(correct_num)
print(train_num)

#plot
import matplotlib.pyplot as plt
# set width of bar
barWidth = 0.25
# Set position of bar on X axis
```

```
r1 = np.arange(len(correct_num))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

# Make the plot
plt.bar(r1, train_num, color='royalblue', width=barWidth, edgecolor='white',␣
 ↪label='Train data')
plt.bar(r2, correct_num, color='green', width=barWidth, edgecolor='white',␣
 ↪label='Correctly classified')
plt.bar(r3, mis_num, color='red', width=barWidth, edgecolor='white',␣
 ↪label='Misclassified')

# Add xticks on the middle of the group bars
plt.title('One Label Articles (6)')
plt.xlabel('Label', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(correct_num))], ["1" ,"2" , "3",␣
 ↪"4" , "5" , "6"])

# Create legend & Show graphic
plt.legend()
plt.savefig('RNN_1.pdf',bbox_inches='tight')
plt.show()
```

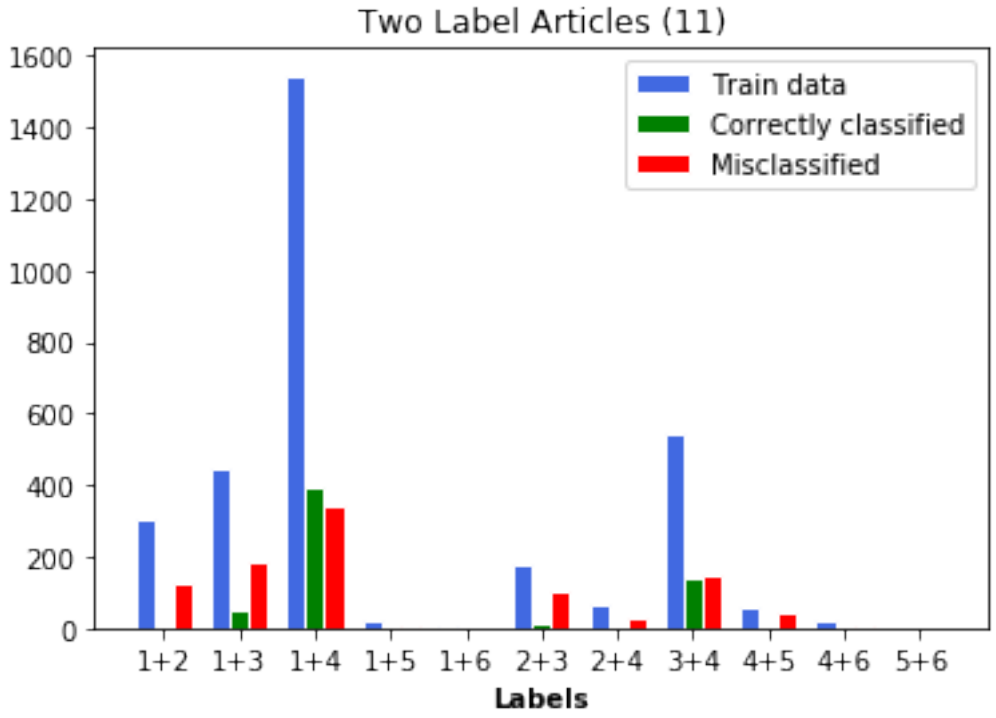
```
[350, 150, 151, 373, 144, 55]
[1240, 1584, 1011, 147, 11, 9]
[3320, 3386, 2448, 1116, 288, 145]
```



[18]:
```
#mis articles that only belong to 2 label.
x = np.array([[1, 1, 0, 0, 0, 0],
              [1, 0, 1, 0, 0, 0],
              [1, 0, 0, 1, 0, 0],
              [1, 0, 0, 0, 1, 0],
              [1, 0, 0, 0, 0, 1],
              [0, 1, 1, 0, 0, 0],
              [0, 1, 0, 1, 0, 0],
              [0, 0, 1, 1, 0, 0],
              [0, 0, 0, 1, 1, 0],
              [0, 0, 0, 1, 0, 1],
              [0, 0, 0, 0, 1, 1]])

c = x.shape[0]
mis_num = []
correct_num = []
train_num = []

for i in range(0,c):
    each_combine = np.repeat(([x[i,:]]), len(mis_y_test), axis=0)
    paired_mis_num = accuracy_score(mis_y_test, each_combine, False)
    mis_num.append(paired_mis_num)

    each_combine = np.repeat(([x[i,:]]), len(correct_y_test), axis=0)
    paired_correct_num = accuracy_score(correct_y_test, each_combine, False)
    correct_num.append(paired_correct_num)

    each_combine = np.repeat(([x[i,:]]), len(y_train), axis=0)
    paired_train_num = accuracy_score(y_train, each_combine, False)
    train_num.append(paired_train_num)

print(mis_num)
print(correct_num)
print(train_num)

#plot
import matplotlib.pyplot as plt
# set width of bar
barWidth = 0.25
# Set position of bar on X axis
r1 = np.arange(len(correct_num))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

# Make the plot
```

```
plt.bar(r1, train_num, color='royalblue', width=barWidth, edgecolor='white',␣
 ↪label='Train data')
plt.bar(r2, correct_num, color='green', width=barWidth, edgecolor='white',␣
 ↪label='Correctly classified')
plt.bar(r3, mis_num, color='red', width=barWidth, edgecolor='white',␣
 ↪label='Misclassified')

# Add xticks on the middle of the group bars
target_labels = ["1+2" ,"1+3" , "1+4", "1+5" , "1+6" , "2+3", "2+4", "3+4",␣
 ↪"4+5",
                  "4+6", "5+6"]
plt.title('Two Label Articles (11)')
plt.xlabel('Labels', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(correct_num))], target_labels)

# Create legend & Show graphic
plt.legend()
plt.savefig('RNN_2.pdf',bbox_inches='tight')
plt.show()
```

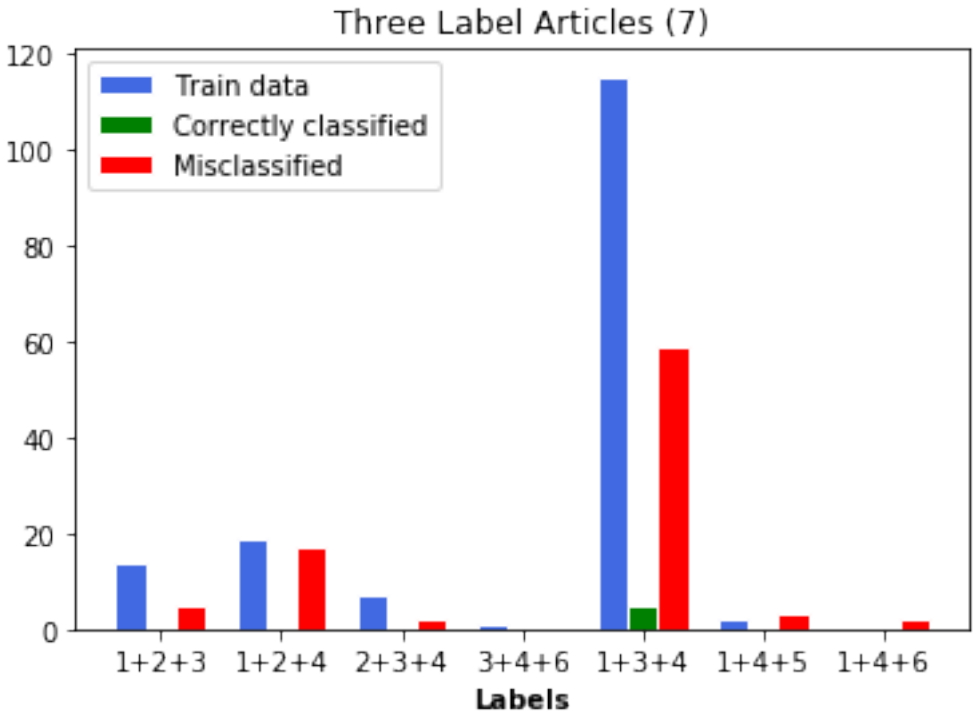
```
[128, 187, 344, 8, 0, 102, 29, 145, 43, 4, 2]
[6, 48, 397, 0, 0, 17, 2, 139, 2, 0, 0]
[303, 447, 1544, 22, 9, 174, 68, 541, 60, 20, 2]
```



[19]:
```
#articles that only belong to 3 label.
x = np.array([[1, 1, 1, 0, 0, 0],
              [1, 1, 0, 1, 0, 0],
              [0, 1, 1, 1, 0, 0],
              [0, 0, 1, 1, 0, 1],
              [1, 0, 1, 1, 0, 0],
              [1, 0, 0, 1, 1, 0],
              [1, 0, 0, 1, 0, 1]])
c = x.shape[0]
mis_num = []
correct_num = []
train_num = []

for i in range(0,c):
    each_combine = np.repeat(([x[i,:]]), len(mis_y_test), axis=0)
    paired_mis_num = accuracy_score(mis_y_test, each_combine, False)
    mis_num.append(paired_mis_num)

    each_combine = np.repeat(([x[i,:]]), len(correct_y_test), axis=0)
    paired_correct_num = accuracy_score(correct_y_test, each_combine, False)
    correct_num.append(paired_correct_num)

    each_combine = np.repeat(([x[i,:]]), len(y_train), axis=0)
    paired_train_num = accuracy_score(y_train, each_combine, False)
    train_num.append(paired_train_num)

print(mis_num)
print(correct_num)
print(train_num)

#plot
import matplotlib.pyplot as plt
# set width of bar
barWidth = 0.25
# Set position of bar on X axis
r1 = np.arange(len(correct_num))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

# Make the plot
plt.bar(r1, train_num, color='royalblue', width=barWidth, edgecolor='white',␣
 ↪label='Train data')
plt.bar(r2, correct_num, color='green', width=barWidth, edgecolor='white',␣
 ↪label='Correctly classified')
plt.bar(r3, mis_num, color='red', width=barWidth, edgecolor='white',␣
 ↪label='Misclassified')
```

```
# Add xticks on the middle of the group bars
target_labels = ["1+2+3" ,"1+2+4" , "2+3+4", "3+4+6" , "1+3+4" , "1+4+5",␣
 ↪"1+4+6"]
plt.title('Three Label Articles (7)')
plt.xlabel('Labels', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(correct_num))], target_labels)

# Create legend & Show graphic
plt.legend()
plt.savefig('RNN_3.pdf',bbox_inches='tight')
plt.show()
```

```
[5, 17, 2, 0, 59, 3, 2]
[0, 0, 0, 0, 5, 0, 0]
[14, 19, 7, 1, 115, 2, 0]
```



[ ]:

[ ]:

[ ]:

0.6 Case

[25]:
```
# get the model
def get_model(n_inputs, n_outputs):
    model = Sequential()
    model.add(Dense(300, input_dim=n_inputs, kernel_initializer='normal',␣
 ↪activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(300, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(n_outputs, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam')
    print(model.summary())
    return model
```

[26]:
```
import time
from tensorflow.keras.layers import Dropout


# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=10, batch_size=500)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")


from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 300)               13900200
_________________________________________________________________
```

```
dropout_1 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_18 (Dense)             (None, 300)               90300
_________________________________________________________________
dropout_2 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_19 (Dense)             (None, 6)                 1806
=================================================================
Total params: 13,992,306
Trainable params: 13,992,306
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
29/29 [==============================] - 25s 845ms/step - loss: 0.5146
Epoch 2/10
29/29 [==============================] - 22s 758ms/step - loss: 0.3203
Epoch 3/10
29/29 [==============================] - 26s 895ms/step - loss: 0.1980
Epoch 4/10
29/29 [==============================] - 25s 851ms/step - loss: 0.1496
Epoch 5/10
29/29 [==============================] - 23s 776ms/step - loss: 0.1170
Epoch 6/10
29/29 [==============================] - 21s 718ms/step - loss: 0.0894
Epoch 7/10
29/29 [==============================] - 23s 781ms/step - loss: 0.0664
Epoch 8/10
29/29 [==============================] - 23s 777ms/step - loss: 0.0477
Epoch 9/10
29/29 [==============================] - 23s 778ms/step - loss: 0.0337
Epoch 10/10
29/29 [==============================] - 24s 826ms/step - loss: 0.0236
--- 307.91 seconds ---
Done
Accuracy =  0.6523623753792804
Fraction Accuracy =  0.9216876173963201
Average Accuracy =  0.92168761739633
```

0.7 case

[27]:
```
# get the model
def get_model(n_inputs, n_outputs):
    model = Sequential()
    model.add(Dense(300, input_dim=n_inputs, kernel_initializer='normal',␣
 ↪activation='relu'))
    #model.add(Dropout(0.3))
```

```
    model.add(Dense(300, activation='relu'))
    #model.add(Dropout(0.3))
    model.add(Dense(n_outputs, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam')
    print(model.summary())
    return model
```

[28]:
```
import time
from tensorflow.keras.layers import Dropout


# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=10, batch_size=500)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")


from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 300)               13900200
_________________________________________________________________
dense_21 (Dense)             (None, 300)               90300
_________________________________________________________________
dense_22 (Dense)             (None, 6)                 1806
=================================================================
Total params: 13,992,306
Trainable params: 13,992,306
Non-trainable params: 0
```

```
_________________________________________________________________
None
Epoch 1/10
29/29 [==============================] - 14s 481ms/step - loss: 0.5037
Epoch 2/10
29/29 [==============================] - 9s 320ms/step - loss: 0.2867
Epoch 3/10
29/29 [==============================] - 11s 384ms/step - loss: 0.1736
Epoch 4/10
29/29 [==============================] - 11s 377ms/step - loss: 0.1234
Epoch 5/10
29/29 [==============================] - 9s 312ms/step - loss: 0.0875
Epoch 6/10
29/29 [==============================] - 12s 403ms/step - loss: 0.0581
Epoch 7/10
29/29 [==============================] - 19s 652ms/step - loss: 0.0355
Epoch 8/10
29/29 [==============================] - 22s 759ms/step - loss: 0.0208
Epoch 9/10
29/29 [==============================] - 23s 791ms/step - loss: 0.0123
Epoch 10/10
29/29 [==============================] - 20s 695ms/step - loss: 0.0077
--- 192.36 seconds ---
Done
Accuracy =  0.6460049125848866
Fraction Accuracy =  0.9193998940422771
Average Accuracy =  0.9193998940422867
```

0.8 case

[29]:
```
# get the model
def get_model(n_inputs, n_outputs):
    model = Sequential()
    model.add(Dense(500, input_dim=n_inputs, kernel_initializer='normal',␣
 ↪activation='relu'))
    #model.add(Dropout(0.3))
    model.add(Dense(300, activation='relu'))
    #model.add(Dropout(0.3))
    model.add(Dense(n_outputs, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam')
    print(model.summary())
    return model
```

[30]:
```
import time
from tensorflow.keras.layers import Dropout
```

```
# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=10, batch_size=500)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")


from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_23 (Dense)             (None, 500)               23167000
_________________________________________________________________
dense_24 (Dense)             (None, 300)               150300
_________________________________________________________________
dense_25 (Dense)             (None, 6)                 1806
=================================================================
Total params: 23,319,106
Trainable params: 23,319,106
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
29/29 [==============================] - 31s 1s/step - loss: 0.4849
Epoch 2/10
29/29 [==============================] - 27s 942ms/step - loss: 0.2531
Epoch 3/10
29/29 [==============================] - 27s 925ms/step - loss: 0.1550
Epoch 4/10
29/29 [==============================] - 27s 928ms/step - loss: 0.1083
Epoch 5/10
```

```
29/29 [==============================] - 27s 943ms/step - loss: 0.0723
Epoch 6/10
29/29 [==============================] - 27s 915ms/step - loss: 0.0437
Epoch 7/10
29/29 [==============================] - 27s 929ms/step - loss: 0.0243
Epoch 8/10
29/29 [==============================] - 25s 877ms/step - loss: 0.0134
Epoch 9/10
29/29 [==============================] - 31s 1s/step - loss: 0.0079
Epoch 10/10
29/29 [==============================] - 27s 929ms/step - loss: 0.0049
--- 334.38 seconds ---
Done
Accuracy =  0.6454269614217598
Fraction Accuracy =  0.9200019265038667
Average Accuracy =  0.9200019265038772
```

0.9 case

[31]:
```
# get the model
def get_model(n_inputs, n_outputs):
    model = Sequential()
    model.add(Dense(200, input_dim=n_inputs, kernel_initializer='normal',␣
 ↪activation='relu'))
    #model.add(Dropout(0.3))
    model.add(Dense(100, activation='relu'))
    #model.add(Dropout(0.3))
    model.add(Dense(n_outputs, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam')
    print(model.summary())
    return model
```

[32]:
```
import time
from tensorflow.keras.layers import Dropout


# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=10, batch_size=500)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")
```

```
from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_26 (Dense)             (None, 200)               9266800
_________________________________________________________________
dense_27 (Dense)             (None, 100)               20100
_________________________________________________________________
dense_28 (Dense)             (None, 6)                 606
=================================================================
Total params: 9,287,506
Trainable params: 9,287,506
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
29/29 [==============================] - 15s 525ms/step - loss: 0.5537
Epoch 2/10
29/29 [==============================] - 12s 427ms/step - loss: 0.3601
Epoch 3/10
29/29 [==============================] - 12s 421ms/step - loss: 0.2250
Epoch 4/10
29/29 [==============================] - 12s 409ms/step - loss: 0.1610
Epoch 5/10
29/29 [==============================] - 12s 411ms/step - loss: 0.1244
Epoch 6/10
29/29 [==============================] - 12s 418ms/step - loss: 0.0962
Epoch 7/10
29/29 [==============================] - 12s 422ms/step - loss: 0.0718
Epoch 8/10
29/29 [==============================] - 12s 416ms/step - loss: 0.0519
Epoch 9/10
29/29 [==============================] - 12s 420ms/step - loss: 0.0365
Epoch 10/10
29/29 [==============================] - 12s 414ms/step - loss: 0.0256
```

```
--- 168.97 seconds ---
Done
Accuracy =  0.6481722294466118
Fraction Accuracy =  0.920507633771602
Average Accuracy =  0.920507633771613
```

[ ]:

[35]:
```python
import time
from tensorflow.keras.layers import Dropout


# define model
model = get_model(n_inputs, n_outputs)
#x_test = x_test.toarray()
# fit model
start_time = time.time()
model.fit(x_train, y_train, verbose=1, epochs=10, batch_size=500)
# make a prediction on the test set
predicted = model.predict(x_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
print("Done")
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_31 (Dense)             (None, 300)               13900200
_________________________________________________________________
dropout_4 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_32 (Dense)             (None, 6)                 1806
=================================================================
Total params: 13,902,006
Trainable params: 13,902,006
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
29/29 [==============================] - 9s 300ms/step - loss: 0.5824
Epoch 2/10
29/29 [==============================] - 7s 232ms/step - loss: 0.3890
Epoch 3/10
29/29 [==============================] - 7s 234ms/step - loss: 0.2797
Epoch 4/10
29/29 [==============================] - 8s 288ms/step - loss: 0.2169
Epoch 5/10
29/29 [==============================] - 9s 295ms/step - loss: 0.1808
```

```
Epoch 6/10
29/29 [==============================] - 7s 237ms/step - loss: 0.1565
Epoch 7/10
29/29 [==============================] - 7s 227ms/step - loss: 0.1369
Epoch 8/10
29/29 [==============================] - 7s 242ms/step - loss: 0.1200
Epoch 9/10
29/29 [==============================] - 7s 245ms/step - loss: 0.1053
Epoch 10/10
29/29 [==============================] - 7s 239ms/step - loss: 0.0924
--- 103.31 seconds ---
Done
```

[ ]:

[ ]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
def get_model(n_inputs, n_outputs):
    model = Sequential()
    model.add(Dense(300, input_dim=n_inputs,
                    kernel_initializer='normal', activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(300, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(n_outputs, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam')
    print(model.summary())
    return model
```

Project_word_embedding

December 15, 2020

[8]:
```
#in terminal install:
#pip install keras
#pip install tensorflow
from numpy import array
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Embedding
```

[2]:
```
corpus = [
    # Positive Reviews

    'This is an excellent movie',
    'The move was fantastic I like it',
    'You should watch it is brilliant',
    'Exceptionally good',
    'Wonderfully directed and executed I like it',
    'Its a fantastic series',
    'Never watched such a brillent movie',
    'It is a Wonderful movie',

    # Negtive Reviews

    "horrible acting",
    'waste of money',
    'pathetic picture',
    'It was very boring',
    'I did not like the movie',
    'The movie was horrible',
    'I will not recommend',
    'The acting is pathetic'
]
sentiments = array([1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0])
```

[3]:
```
word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(corpus)
```

[4]:
```
vocab_length = len(word_tokenizer.word_index) + 1
vocab_length
```

[4]:
```
44
```

[6]:
```
embedded_sentences = word_tokenizer.texts_to_sequences(corpus)
print(embedded_sentences)
```

```
[[14, 3, 15, 16, 1], [4, 17, 6, 9, 5, 7, 2], [18, 19, 20, 2, 3, 21], [22, 23],
[24, 25, 26, 27, 5, 7, 2], [28, 8, 9, 29], [30, 31, 32, 8, 33, 1], [2, 3, 8, 34,
1], [10, 11], [35, 36, 37], [12, 38], [2, 6, 39, 40], [5, 41, 13, 7, 4, 1], [4,
1, 6, 10], [5, 42, 13, 43], [4, 11, 3, 12]]
```

[7]:
```
from nltk.tokenize import word_tokenize

word_count = lambda sentence: len(word_tokenize(sentence))
longest_sentence = max(corpus, key=word_count)
length_long_sentence = len(word_tokenize(longest_sentence))

padded_sentences = pad_sequences(embedded_sentences, length_long_sentence,␣
 ↪padding='post')

print(padded_sentences)
```

```
[[14  3 15 16  1  0  0]
 [ 4 17  6  9  5  7  2]
 [18 19 20  2  3 21  0]
 [22 23  0  0  0  0  0]
 [24 25 26 27  5  7  2]
 [28  8  9 29  0  0  0]
 [30 31 32  8 33  1  0]
 [ 2  3  8 34  1  0  0]
 [10 11  0  0  0  0  0]
 [35 36 37  0  0  0  0]
 [12 38  0  0  0  0  0]
 [ 2  6 39 40  0  0  0]
 [ 5 41 13  7  4  1  0]
 [ 4  1  6 10  0  0  0]
 [ 5 42 13 43  0  0  0]
 [ 4 11  3 12  0  0  0]]
```

[5]:
```
from numpy import array
from numpy import asarray
from numpy import zeros
```

```
embeddings_dictionary = dict()
glove_file = open('/Users/serena/Desktop/Statistics/2020Fall/251/Project/
 ↪wordEmbedding/glove.6B.100d.txt', encoding="utf8")
```

[10]:
```
for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary [word] = vector_dimensions

glove_file.close()
```

[11]:
```
embedding_matrix = zeros((vocab_length, 100))
for word, index in word_tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

[13]:
```
embedding_matrix.shape
```

[13]:
```
(44, 100)
```

[16]:
```
model = Sequential()
embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix],
                            input_length=length_long_sentence, trainable=False)
model.add(embedding_layer)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

[17]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 7, 100)            4400
_________________________________________________________________
flatten (Flatten)            (None, 700)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 701
=================================================================
Total params: 5,101
Trainable params: 701
Non-trainable params: 4,400
_________________________________________________________________
```

None

[18]:
```
model.fit(padded_sentences, sentiments, epochs=100, verbose=1)
```

```
Epoch 1/100
1/1 [==============================] - 0s 6ms/step - loss: 0.7085 - acc: 0.5000
Epoch 2/100
1/1 [==============================] - 0s 1ms/step - loss: 0.6819 - acc: 0.5625
Epoch 3/100
1/1 [==============================] - 0s 3ms/step - loss: 0.6568 - acc: 0.5625
Epoch 4/100
1/1 [==============================] - 0s 10ms/step - loss: 0.6328 - acc: 0.6250
Epoch 5/100
1/1 [==============================] - 0s 4ms/step - loss: 0.6100 - acc: 0.6250
Epoch 6/100
1/1 [==============================] - 0s 1ms/step - loss: 0.5881 - acc: 0.7500
Epoch 7/100
1/1 [==============================] - 0s 5ms/step - loss: 0.5672 - acc: 0.8125
Epoch 8/100
1/1 [==============================] - 0s 4ms/step - loss: 0.5471 - acc: 0.8125
Epoch 9/100
1/1 [==============================] - 0s 1ms/step - loss: 0.5278 - acc: 0.8750
Epoch 10/100
1/1 [==============================] - 0s 4ms/step - loss: 0.5093 - acc: 0.8750
Epoch 11/100
1/1 [==============================] - 0s 2ms/step - loss: 0.4915 - acc: 0.9375
Epoch 12/100
1/1 [==============================] - 0s 1ms/step - loss: 0.4743 - acc: 0.9375
Epoch 13/100
1/1 [==============================] - 0s 2ms/step - loss: 0.4579 - acc: 1.0000
Epoch 14/100
1/1 [==============================] - 0s 1ms/step - loss: 0.4421 - acc: 1.0000
Epoch 15/100
1/1 [==============================] - 0s 2ms/step - loss: 0.4270 - acc: 1.0000
Epoch 16/100
1/1 [==============================] - 0s 1ms/step - loss: 0.4125 - acc: 1.0000
Epoch 17/100
1/1 [==============================] - 0s 2ms/step - loss: 0.3987 - acc: 1.0000
Epoch 18/100
1/1 [==============================] - 0s 1ms/step - loss: 0.3854 - acc: 1.0000
Epoch 19/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3727 - acc: 1.0000
Epoch 20/100
1/1 [==============================] - 0s 2ms/step - loss: 0.3605 - acc: 1.0000
Epoch 21/100
1/1 [==============================] - ETA: 0s - loss: 0.3489 - acc: 1.000 - 0s
1ms/step - loss: 0.3489 - acc: 1.0000
Epoch 22/100
```

```
1/1 [==============================] - 0s 1ms/step - loss: 0.3377 - acc: 1.0000
Epoch 23/100
1/1 [==============================] - 0s 1ms/step - loss: 0.3270 - acc: 1.0000
Epoch 24/100
1/1 [==============================] - 0s 3ms/step - loss: 0.3168 - acc: 1.0000
Epoch 25/100
1/1 [==============================] - 0s 2ms/step - loss: 0.3069 - acc: 1.0000
Epoch 26/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2975 - acc: 1.0000
Epoch 27/100
1/1 [==============================] - 0s 1ms/step - loss: 0.2884 - acc: 1.0000
Epoch 28/100
1/1 [==============================] - 0s 1ms/step - loss: 0.2798 - acc: 1.0000
Epoch 29/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2715 - acc: 1.0000
Epoch 30/100
1/1 [==============================] - 0s 1ms/step - loss: 0.2635 - acc: 1.0000
Epoch 31/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2559 - acc: 1.0000
Epoch 32/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2485 - acc: 1.0000
Epoch 33/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2415 - acc: 1.0000
Epoch 34/100
1/1 [==============================] - 0s 1ms/step - loss: 0.2348 - acc: 1.0000
Epoch 35/100
1/1 [==============================] - 0s 1ms/step - loss: 0.2283 - acc: 1.0000
Epoch 36/100
1/1 [==============================] - 0s 1ms/step - loss: 0.2221 - acc: 1.0000
Epoch 37/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2162 - acc: 1.0000
Epoch 38/100
1/1 [==============================] - 0s 2ms/step - loss: 0.2105 - acc: 1.0000
Epoch 39/100
1/1 [==============================] - 0s 3ms/step - loss: 0.2050 - acc: 1.0000
Epoch 40/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1997 - acc: 1.0000
Epoch 41/100
1/1 [==============================] - 0s 1ms/step - loss: 0.1947 - acc: 1.0000
Epoch 42/100
1/1 [==============================] - 0s 3ms/step - loss: 0.1898 - acc: 1.0000
Epoch 43/100
1/1 [==============================] - 0s 1ms/step - loss: 0.1851 - acc: 1.0000
Epoch 44/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1806 - acc: 1.0000
Epoch 45/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1763 - acc: 1.0000
Epoch 46/100
```

```
1/1 [==============================] - 0s 1ms/step - loss: 0.1721 - acc: 1.0000
Epoch 47/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1681 - acc: 1.0000
Epoch 48/100
1/1 [==============================] - 0s 1ms/step - loss: 0.1642 - acc: 1.0000
Epoch 49/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1605 - acc: 1.0000
Epoch 50/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1569 - acc: 1.0000
Epoch 51/100
1/1 [==============================] - 0s 1ms/step - loss: 0.1534 - acc: 1.0000
Epoch 52/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1500 - acc: 1.0000
Epoch 53/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1468 - acc: 1.0000
Epoch 54/100
1/1 [==============================] - 0s 7ms/step - loss: 0.1437 - acc: 1.0000
Epoch 55/100
1/1 [==============================] - 0s 10ms/step - loss: 0.1406 - acc: 1.0000
Epoch 56/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1377 - acc: 1.0000
Epoch 57/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1349 - acc: 1.0000
Epoch 58/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1322 - acc: 1.0000
Epoch 59/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1295 - acc: 1.0000
Epoch 60/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1270 - acc: 1.0000
Epoch 61/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1245 - acc: 1.0000
Epoch 62/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1221 - acc: 1.0000
Epoch 63/100
1/1 [==============================] - 0s 1ms/step - loss: 0.1198 - acc: 1.0000
Epoch 64/100
1/1 [==============================] - 0s 3ms/step - loss: 0.1175 - acc: 1.0000
Epoch 65/100
1/1 [==============================] - 0s 8ms/step - loss: 0.1154 - acc: 1.0000
Epoch 66/100
1/1 [==============================] - 0s 1ms/step - loss: 0.1133 - acc: 1.0000
Epoch 67/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1112 - acc: 1.0000
Epoch 68/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1092 - acc: 1.0000
Epoch 69/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1073 - acc: 1.0000
Epoch 70/100
```

```
1/1 [==============================] - 0s 2ms/step - loss: 0.1054 - acc: 1.0000
Epoch 71/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1036 - acc: 1.0000
Epoch 72/100
1/1 [==============================] - 0s 1ms/step - loss: 0.1018 - acc: 1.0000
Epoch 73/100
1/1 [==============================] - 0s 2ms/step - loss: 0.1001 - acc: 1.0000
Epoch 74/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0984 - acc: 1.0000
Epoch 75/100
1/1 [==============================] - 0s 3ms/step - loss: 0.0968 - acc: 1.0000
Epoch 76/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0952 - acc: 1.0000
Epoch 77/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0937 - acc: 1.0000
Epoch 78/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0922 - acc: 1.0000
Epoch 79/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0907 - acc: 1.0000
Epoch 80/100
1/1 [==============================] - 0s 4ms/step - loss: 0.0893 - acc: 1.0000
Epoch 81/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0879 - acc: 1.0000
Epoch 82/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0866 - acc: 1.0000
Epoch 83/100
1/1 [==============================] - 0s 1ms/step - loss: 0.0853 - acc: 1.0000
Epoch 84/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0840 - acc: 1.0000
Epoch 85/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0827 - acc: 1.0000
Epoch 86/100
1/1 [==============================] - 0s 3ms/step - loss: 0.0815 - acc: 1.0000
Epoch 87/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0803 - acc: 1.0000
Epoch 88/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0792 - acc: 1.0000
Epoch 89/100
1/1 [==============================] - 0s 3ms/step - loss: 0.0780 - acc: 1.0000
Epoch 90/100
1/1 [==============================] - 0s 1ms/step - loss: 0.0769 - acc: 1.0000
Epoch 91/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0758 - acc: 1.0000
Epoch 92/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0748 - acc: 1.0000
Epoch 93/100
1/1 [==============================] - 0s 1ms/step - loss: 0.0737 - acc: 1.0000
Epoch 94/100
```

```
1/1 [==============================] - 0s 2ms/step - loss: 0.0727 - acc: 1.0000
Epoch 95/100
1/1 [==============================] - 0s 918us/step - loss: 0.0718 - acc:
1.0000
Epoch 96/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0708 - acc: 1.0000
Epoch 97/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0698 - acc: 1.0000
Epoch 98/100
1/1 [==============================] - 0s 1ms/step - loss: 0.0689 - acc: 1.0000
Epoch 99/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0680 - acc: 1.0000
Epoch 100/100
1/1 [==============================] - 0s 2ms/step - loss: 0.0671 - acc: 1.0000
```

[18]:
```
<tensorflow.python.keras.callbacks.History at 0x1a40c53050>
```

0.1 keras LSTM

[1]:
```
#import data set
import pandas as pd
import numpy as np
import re
import string

test = pd.read_csv('data/NLP_on_Research_Articles/archive/test.csv')
train = pd.read_csv('data/NLP_on_Research_Articles/archive/train.csv')

print(train.shape)
print(test.shape)
```

```
(20972, 9)
(8989, 3)
```

[2]:
```
#deal with the column names
train.columns = train.columns.str.strip().str.lower().str.replace(' ', '_').str.
 ↪replace('(', '').str.replace(')', '')
print('Train Data shape: ', train.shape)
```

```
Train Data shape:  (20972, 9)
```

[3]:
```
train['description'] = train['title'] + ". " + train['abstract']
train['description'] = train['description'].str.lower()
```

[4]:
```
def preprocess_text(sen):
    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sen)
```

```
    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

    # Removing multiple spaces
    #sentence = re.sub(r'\s+', ' ', sentence)

    return sentence
```

[5]:
```
X = []
sentences = list(train["description"])
for sen in sentences:
    X.append(preprocess_text(sen))

#print(X.shape)
y = np.asarray(train[train.columns[3:9]].values)
#print(y.shape)
```

[6]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
↪random_state=42)
```

[9]:
```
#tokenizer = Tokenizer(num_words=5000)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)

X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

vocab_size = len(tokenizer.word_index) + 1

maxlen = 200

X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

[10]:
```
print(vocab_size)
print(X_train.shape)
print(X_test.shape)
```

```
41944
(14051, 200)
(6921, 200)
```

[11]:
```
from numpy import array
from numpy import asarray
```

```
from numpy import zeros

embeddings_dictionary = dict()

glove_file = open('/Users/serena/Desktop/Statistics/2020Fall/251/Project/
 ↪wordEmbedding/glove.6B.100d.txt',
                  encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary[word] = vector_dimensions
glove_file.close()

embedding_matrix = zeros((vocab_size, 100))
for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

[17]:
```
embedding_matrix
```

[17]:
```
array([[ 0.        ,  0.        ,  0.        , …,  0.        ,
         0.        ,  0.        ],
       [-0.038194  , -0.24487001,  0.72812003, …, -0.1459    ,
         0.82779998,  0.27061999],
       [-0.1529    , -0.24279   ,  0.89837003, …, -0.59100002,
         1.00390005,  0.20664001],
       …,
       [ 0.        ,  0.        ,  0.        , …,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , …,  0.        ,
         0.        ,  0.        ],
       [-0.34408   , -0.060351  ,  0.064278  , …,  0.37685001,
         0.054019  ,  0.30603999]])
```

0.2 case1

[19]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model
deep_inputs = Input(shape=(maxlen,))
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix],␣
 ↪trainable=False)(deep_inputs)
LSTM_Layer_1 = LSTM(128)(embedding_layer)
dense_layer_1 = Dense(6, activation='sigmoid')(LSTM_Layer_1)
```

```
model = Model(inputs=deep_inputs, outputs=dense_layer_1)

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

[20]:
```
print(model.summary())
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 200)]             0
_________________________________________________________________
embedding (Embedding)        (None, 200, 100)          4194400
_________________________________________________________________
lstm (LSTM)                  (None, 128)               117248
_________________________________________________________________
dense (Dense)                (None, 6)                 774
=================================================================
Total params: 4,312,422
Trainable params: 118,022
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[22]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=128, epochs=10, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
```

```
Epoch 1/10
88/88 [==============================] - 48s 546ms/step - loss: 0.4447 - acc:
0.4573 - val_loss: 0.4122 - val_acc: 0.4980
Epoch 2/10
88/88 [==============================] - 48s 550ms/step - loss: 0.3896 - acc:
0.5450 - val_loss: 0.3686 - val_acc: 0.5802
Epoch 3/10
88/88 [==============================] - 43s 487ms/step - loss: 0.3627 - acc:
0.6170 - val_loss: 0.3676 - val_acc: 0.5820
Epoch 4/10
88/88 [==============================] - 43s 494ms/step - loss: 0.3347 - acc:
0.6723 - val_loss: 0.3102 - val_acc: 0.7022
Epoch 5/10
88/88 [==============================] - 43s 491ms/step - loss: 0.2935 - acc:
0.7254 - val_loss: 0.2853 - val_acc: 0.7296
```

```
Epoch 6/10
88/88 [==============================] - 43s 489ms/step - loss: 0.3768 - acc:
0.5578 - val_loss: 0.3690 - val_acc: 0.5607
Epoch 7/10
88/88 [==============================] - 43s 484ms/step - loss: 0.3567 - acc:
0.6051 - val_loss: 0.3130 - val_acc: 0.7332
Epoch 8/10
88/88 [==============================] - 44s 503ms/step - loss: 0.3320 - acc:
0.6970 - val_loss: 0.3024 - val_acc: 0.7360
Epoch 9/10
88/88 [==============================] - 43s 490ms/step - loss: 0.2895 - acc:
0.7340 - val_loss: 0.2727 - val_acc: 0.7488
Epoch 10/10
88/88 [==============================] - 45s 512ms/step - loss: 0.2697 - acc:
0.7447 - val_loss: 0.2642 - val_acc: 0.7503
--- 468.64 seconds ---
```

[23]:
```
from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Accuracy =  0.560323652651351
Fraction Accuracy =  0.8852526128208794
Average Accuracy =  0.8852526128208833
```

0.3 Case2

[25]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model
deep_inputs = Input(shape=(maxlen,))
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix],␣
 ↪trainable=False)(deep_inputs)
LSTM_Layer_1 = LSTM(128)(embedding_layer)
dense_layer_1 = Dense(100, activation='relu')(LSTM_Layer_1)
dense_layer_2 = Dense(6, activation='sigmoid')(dense_layer_1)
model = Model(inputs=deep_inputs, outputs=dense_layer_2)

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "functional_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 200)]             0
_________________________________________________________________
embedding_2 (Embedding)      (None, 200, 100)          4194400
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               117248
_________________________________________________________________
dense_2 (Dense)              (None, 100)               12900
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 606
=================================================================
Total params: 4,325,154
Trainable params: 130,754
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[26]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=128, epochs=10, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/10
88/88 [==============================] - 44s 501ms/step - loss: 0.4462 - acc:
0.4308 - val_loss: 0.4095 - val_acc: 0.4995
Epoch 2/10
88/88 [==============================] - 45s 508ms/step - loss: 0.3691 - acc:
0.5934 - val_loss: 0.3202 - val_acc: 0.6756
Epoch 3/10
88/88 [==============================] - 43s 493ms/step - loss: 0.2981 - acc:
0.7184 - val_loss: 0.2837 - val_acc: 0.7286
```

```
Epoch 4/10
88/88 [==============================] - 42s 478ms/step - loss: 0.2737 - acc:
0.7351 - val_loss: 0.2720 - val_acc: 0.7090
Epoch 5/10
88/88 [==============================] - 41s 471ms/step - loss: 0.2588 - acc:
0.7415 - val_loss: 0.2546 - val_acc: 0.7296
Epoch 6/10
88/88 [==============================] - 41s 471ms/step - loss: 0.2497 - acc:
0.7385 - val_loss: 0.2491 - val_acc: 0.7453
Epoch 7/10
88/88 [==============================] - 42s 472ms/step - loss: 0.2380 - acc:
0.7383 - val_loss: 0.2345 - val_acc: 0.7389
Epoch 8/10
88/88 [==============================] - 42s 473ms/step - loss: 0.2260 - acc:
0.7418 - val_loss: 0.2245 - val_acc: 0.7552
Epoch 9/10
88/88 [==============================] - 43s 484ms/step - loss: 0.2198 - acc:
0.7459 - val_loss: 0.2244 - val_acc: 0.7318
Epoch 10/10
88/88 [==============================] - 45s 516ms/step - loss: 0.2078 - acc:
0.7515 - val_loss: 0.2175 - val_acc: 0.7645
--- 451.81 seconds ---
Accuracy =  0.6191301834994943
Fraction Accuracy =  0.908828204016756
Average Accuracy =  0.9088282040167606
```

0.4 case3

[27]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model
deep_inputs = Input(shape=(maxlen,))
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix],␣
 ↪trainable=False)(deep_inputs)
LSTM_Layer_1 = LSTM(128)(embedding_layer)
dense_layer_1 = Dense(100, activation='relu')(LSTM_Layer_1)
dense_layer_2 = Dense(100, activation='relu')(LSTM_Layer_1)
dense_layer_3 = Dense(6, activation='sigmoid')(dense_layer_2)
model = Model(inputs=deep_inputs, outputs=dense_layer_3)

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "functional_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 200)]             0
```

```
_________________________________________________________________
embedding_3 (Embedding)      (None, 200, 100)          4194400
_________________________________________________________________
lstm_3 (LSTM)                (None, 128)               117248
_________________________________________________________________
dense_5 (Dense)              (None, 100)               12900
_________________________________________________________________
dense_6 (Dense)              (None, 6)                 606
=================================================================
Total params: 4,325,154
Trainable params: 130,754
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[28]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=200, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
57/57 [==============================] - 42s 732ms/step - loss: 0.4641 - acc:
0.4168 - val_loss: 0.4193 - val_acc: 0.4593
Epoch 2/20
57/57 [==============================] - 39s 684ms/step - loss: 0.4040 - acc:
0.5225 - val_loss: 0.4160 - val_acc: 0.4888
Epoch 3/20
57/57 [==============================] - 41s 727ms/step - loss: 0.3904 - acc:
0.5377 - val_loss: 0.3696 - val_acc: 0.6055
Epoch 4/20
57/57 [==============================] - 41s 718ms/step - loss: 0.3222 - acc:
0.6943 - val_loss: 0.3006 - val_acc: 0.7197
Epoch 5/20
57/57 [==============================] - 38s 666ms/step - loss: 0.2873 - acc:
```

```
0.7277 - val_loss: 0.2842 - val_acc: 0.7250
Epoch 6/20
57/57 [==============================] - 40s 699ms/step - loss: 0.2717 - acc:
0.7414 - val_loss: 0.2726 - val_acc: 0.7385
Epoch 7/20
57/57 [==============================] - 40s 697ms/step - loss: 0.2614 - acc:
0.7445 - val_loss: 0.2657 - val_acc: 0.7453
Epoch 8/20
57/57 [==============================] - 39s 692ms/step - loss: 0.2558 - acc:
0.7403 - val_loss: 0.2562 - val_acc: 0.7318
Epoch 9/20
57/57 [==============================] - 38s 673ms/step - loss: 0.2481 - acc:
0.7410 - val_loss: 0.2467 - val_acc: 0.7506
Epoch 10/20
57/57 [==============================] - 40s 695ms/step - loss: 0.2404 - acc:
0.7429 - val_loss: 0.2462 - val_acc: 0.7467
Epoch 11/20
57/57 [==============================] - 42s 743ms/step - loss: 0.2308 - acc:
0.7464 - val_loss: 0.2343 - val_acc: 0.7065
Epoch 12/20
57/57 [==============================] - 41s 721ms/step - loss: 0.2245 - acc:
0.7455 - val_loss: 0.2305 - val_acc: 0.7421
Epoch 13/20
57/57 [==============================] - 41s 720ms/step - loss: 0.2193 - acc:
0.7508 - val_loss: 0.2256 - val_acc: 0.7613
Epoch 14/20
57/57 [==============================] - 46s 803ms/step - loss: 0.2120 - acc:
0.7525 - val_loss: 0.2214 - val_acc: 0.7328
Epoch 15/20
57/57 [==============================] - 41s 727ms/step - loss: 0.2082 - acc:
0.7515 - val_loss: 0.2220 - val_acc: 0.7218
Epoch 16/20
57/57 [==============================] - 40s 704ms/step - loss: 0.2031 - acc:
0.7538 - val_loss: 0.2312 - val_acc: 0.7296
Epoch 17/20
57/57 [==============================] - 41s 722ms/step - loss: 0.2013 - acc:
0.7557 - val_loss: 0.2148 - val_acc: 0.7691
Epoch 18/20
57/57 [==============================] - 39s 691ms/step - loss: 0.1951 - acc:
0.7584 - val_loss: 0.2262 - val_acc: 0.7275
Epoch 19/20
57/57 [==============================] - 39s 684ms/step - loss: 0.1905 - acc:
0.7618 - val_loss: 0.2167 - val_acc: 0.7012
Epoch 20/20
57/57 [==============================] - 41s 713ms/step - loss: 0.1891 - acc:
0.7586 - val_loss: 0.2274 - val_acc: 0.7382
--- 845.78 seconds ---
Accuracy =  0.6045369166305448
```

```
Fraction Accuracy =  0.9039397004286387
Average Accuracy =  0.903939700428647
```

0.5 case4

[29]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model
deep_inputs = Input(shape=(maxlen,))
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix],␣
 ↪trainable=False)(deep_inputs)
LSTM_Layer_1 = LSTM(128)(embedding_layer)
dense_layer_1 = Dense(100, activation='relu')(LSTM_Layer_1)
dense_layer_2 = Dense(6, activation='sigmoid')(dense_layer_1)
model = Model(inputs=deep_inputs, outputs=dense_layer_2)

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "functional_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 200)]             0
_________________________________________________________________
embedding_4 (Embedding)      (None, 200, 100)          4194400
_________________________________________________________________
lstm_4 (LSTM)                (None, 128)               117248
_________________________________________________________________
dense_7 (Dense)              (None, 100)               12900
_________________________________________________________________
dense_8 (Dense)              (None, 6)                 606
=================================================================
Total params: 4,325,154
Trainable params: 130,754
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[30]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))
```

```
from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
88/88 [==============================] - 45s 506ms/step - loss: 0.4450 - acc:
0.4620 - val_loss: 0.4005 - val_acc: 0.5247
Epoch 2/20
88/88 [==============================] - 43s 493ms/step - loss: 0.3913 - acc:
0.5484 - val_loss: 0.3865 - val_acc: 0.5560
Epoch 3/20
88/88 [==============================] - 46s 528ms/step - loss: 0.3594 - acc:
0.6071 - val_loss: 0.3035 - val_acc: 0.7090
Epoch 4/20
88/88 [==============================] - 44s 499ms/step - loss: 0.3062 - acc:
0.7060 - val_loss: 0.3850 - val_acc: 0.5834
Epoch 5/20
88/88 [==============================] - 48s 544ms/step - loss: 0.2965 - acc:
0.7103 - val_loss: 0.2805 - val_acc: 0.7250
Epoch 6/20
88/88 [==============================] - 47s 534ms/step - loss: 0.2699 - acc:
0.7210 - val_loss: 0.2617 - val_acc: 0.7158
Epoch 7/20
88/88 [==============================] - 47s 539ms/step - loss: 0.2581 - acc:
0.7193 - val_loss: 0.2555 - val_acc: 0.7150
Epoch 8/20
88/88 [==============================] - 42s 483ms/step - loss: 0.2494 - acc:
0.7252 - val_loss: 0.2480 - val_acc: 0.7257
Epoch 9/20
88/88 [==============================] - 43s 491ms/step - loss: 0.2395 - acc:
0.7335 - val_loss: 0.2390 - val_acc: 0.7432
Epoch 10/20
88/88 [==============================] - 50s 567ms/step - loss: 0.2314 - acc:
0.7413 - val_loss: 0.2346 - val_acc: 0.7261
Epoch 11/20
88/88 [==============================] - 45s 512ms/step - loss: 0.2239 - acc:
0.7412 - val_loss: 0.2281 - val_acc: 0.7531
Epoch 12/20
88/88 [==============================] - 45s 511ms/step - loss: 0.2154 - acc:
0.7483 - val_loss: 0.2233 - val_acc: 0.7289
```

```
Epoch 13/20
88/88 [==============================] - 36s 413ms/step - loss: 0.2087 - acc:
0.7486 - val_loss: 0.2187 - val_acc: 0.7364
Epoch 14/20
88/88 [==============================] - 37s 421ms/step - loss: 0.2029 - acc:
0.7537 - val_loss: 0.2168 - val_acc: 0.7446
Epoch 15/20
88/88 [==============================] - 36s 407ms/step - loss: 0.1963 - acc:
0.7569 - val_loss: 0.2111 - val_acc: 0.7545
Epoch 16/20
88/88 [==============================] - 37s 415ms/step - loss: 0.1915 - acc:
0.7565 - val_loss: 0.2079 - val_acc: 0.7652
Epoch 17/20
88/88 [==============================] - 37s 415ms/step - loss: 0.1866 - acc:
0.7627 - val_loss: 0.2132 - val_acc: 0.7239
Epoch 18/20
88/88 [==============================] - 38s 427ms/step - loss: 0.1815 - acc:
0.7641 - val_loss: 0.2106 - val_acc: 0.7617
Epoch 19/20
88/88 [==============================] - 39s 441ms/step - loss: 0.1777 - acc:
0.7684 - val_loss: 0.2091 - val_acc: 0.7378
Epoch 20/20
88/88 [==============================] - 37s 421ms/step - loss: 0.1720 - acc:
0.7704 - val_loss: 0.2085 - val_acc: 0.7367
--- 869.86 seconds ---
Accuracy =  0.6266435486201416
Fraction Accuracy =  0.9113085777585042
Average Accuracy =  0.9113085777585127
```

0.6 case5

[36]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model

model = Sequential()
model.add(Embedding(vocab_size,
                    100,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False))
#model.add(SpatialDropout1D(0.3))
#model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3))
model.add(LSTM(128))
model.add(Dense(100, activation='relu'))
#model.add(Dropout(0.8))
```

```
#model.add(Dense(100, activation='relu'))
#model.add(Dropout(0.8))
model.add(Dense(6, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 200, 100)          4194400
_________________________________________________________________
lstm_8 (LSTM)                (None, 128)               117248
_________________________________________________________________
dense_15 (Dense)             (None, 100)               12900
_________________________________________________________________
dense_16 (Dense)             (None, 6)                 606
=================================================================
Total params: 4,325,154
Trainable params: 130,754
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[37]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
88/88 [==============================] - 38s 433ms/step - loss: 0.4491 - acc:
0.4381 - val_loss: 0.3979 - val_acc: 0.5336
Epoch 2/20
```

```
88/88 [==============================] - 43s 489ms/step - loss: 0.4081 - acc:
0.4927 - val_loss: 0.4037 - val_acc: 0.5212
Epoch 3/20
88/88 [==============================] - 52s 585ms/step - loss: 0.3848 - acc:
0.5575 - val_loss: 0.3559 - val_acc: 0.6631
Epoch 4/20
88/88 [==============================] - 46s 518ms/step - loss: 0.3104 - acc:
0.7013 - val_loss: 0.2785 - val_acc: 0.7392
Epoch 5/20
88/88 [==============================] - 45s 511ms/step - loss: 0.2892 - acc:
0.7186 - val_loss: 0.2975 - val_acc: 0.6998
Epoch 6/20
88/88 [==============================] - 45s 510ms/step - loss: 0.2736 - acc:
0.7278 - val_loss: 0.2700 - val_acc: 0.6933
Epoch 7/20
88/88 [==============================] - 44s 499ms/step - loss: 0.2584 - acc:
0.7324 - val_loss: 0.2570 - val_acc: 0.7410
Epoch 8/20
88/88 [==============================] - 44s 497ms/step - loss: 0.2495 - acc:
0.7258 - val_loss: 0.2483 - val_acc: 0.7069
Epoch 9/20
88/88 [==============================] - 44s 499ms/step - loss: 0.2402 - acc:
0.7320 - val_loss: 0.2401 - val_acc: 0.7392
Epoch 10/20
88/88 [==============================] - 44s 498ms/step - loss: 0.2306 - acc:
0.7521 - val_loss: 0.2408 - val_acc: 0.7407
Epoch 11/20
88/88 [==============================] - 44s 503ms/step - loss: 0.2232 - acc:
0.7512 - val_loss: 0.2284 - val_acc: 0.7503
Epoch 12/20
88/88 [==============================] - 43s 491ms/step - loss: 0.2132 - acc:
0.7560 - val_loss: 0.2203 - val_acc: 0.7702
Epoch 13/20
88/88 [==============================] - 34s 385ms/step - loss: 0.2048 - acc:
0.7586 - val_loss: 0.2210 - val_acc: 0.7126
Epoch 14/20
88/88 [==============================] - 34s 391ms/step - loss: 0.2029 - acc:
0.7603 - val_loss: 0.2145 - val_acc: 0.7432
Epoch 15/20
88/88 [==============================] - 34s 390ms/step - loss: 0.1947 - acc:
0.7644 - val_loss: 0.2160 - val_acc: 0.7581
Epoch 16/20
88/88 [==============================] - 34s 386ms/step - loss: 0.1912 - acc:
0.7625 - val_loss: 0.2147 - val_acc: 0.7417
Epoch 17/20
88/88 [==============================] - 34s 391ms/step - loss: 0.1854 - acc:
0.7662 - val_loss: 0.2131 - val_acc: 0.7229
Epoch 18/20
```

```
88/88 [==============================] - 35s 402ms/step - loss: 0.1817 - acc:
0.7673 - val_loss: 0.2098 - val_acc: 0.7478
Epoch 19/20
88/88 [==============================] - 34s 386ms/step - loss: 0.1773 - acc:
0.7693 - val_loss: 0.2106 - val_acc: 0.7503
Epoch 20/20
88/88 [==============================] - 34s 391ms/step - loss: 0.1758 - acc:
0.7681 - val_loss: 0.2135 - val_acc: 0.7634
--- 834.84 seconds ---
Accuracy =  0.6306892067620286
Fraction Accuracy =  0.9114289842508235
Average Accuracy =  0.9114289842508309
```

0.7 case6

[38]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model

model = Sequential()
model.add(Embedding(vocab_size,
                    100,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False))
#model.add(SpatialDropout1D(0.3))
#model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3))
model.add(LSTM(128))
model.add(Dense(100, activation='relu'))
#model.add(Dropout(0.8))
#model.add(Dense(100, activation='relu'))
#model.add(Dropout(0.8))
model.add(Dense(6, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 200, 100)          4194400
_________________________________________________________________
lstm_9 (LSTM)                (None, 128)               117248
_________________________________________________________________
dense_17 (Dense)             (None, 100)               12900
_________________________________________________________________
dense_18 (Dense)             (None, 6)                 606
```

```
=================================================================
Total params: 4,325,154
Trainable params: 130,754
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[39]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=200, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
57/57 [==============================] - 37s 647ms/step - loss: 0.4649 - acc:
0.3977 - val_loss: 0.4281 - val_acc: 0.4425
Epoch 2/20
57/57 [==============================] - 40s 703ms/step - loss: 0.4236 - acc:
0.4492 - val_loss: 0.4211 - val_acc: 0.4112
Epoch 3/20
57/57 [==============================] - 46s 804ms/step - loss: 0.4003 - acc:
0.5191 - val_loss: 0.3413 - val_acc: 0.6827
Epoch 4/20
57/57 [==============================] - 43s 747ms/step - loss: 0.3488 - acc:
0.6520 - val_loss: 0.3436 - val_acc: 0.6485
Epoch 5/20
57/57 [==============================] - 43s 749ms/step - loss: 0.3027 - acc:
0.7181 - val_loss: 0.2841 - val_acc: 0.7360
Epoch 6/20
57/57 [==============================] - 43s 753ms/step - loss: 0.2706 - acc:
0.7426 - val_loss: 0.2669 - val_acc: 0.7421
Epoch 7/20
57/57 [==============================] - 42s 744ms/step - loss: 0.2632 - acc:
0.7426 - val_loss: 0.2581 - val_acc: 0.7492
Epoch 8/20
```

```
57/57 [==============================] - 43s 751ms/step - loss: 0.2496 - acc:
0.7460 - val_loss: 0.2496 - val_acc: 0.7407
Epoch 9/20
57/57 [==============================] - 42s 741ms/step - loss: 0.2404 - acc:
0.7522 - val_loss: 0.2464 - val_acc: 0.7264
Epoch 10/20
57/57 [==============================] - 42s 741ms/step - loss: 0.2323 - acc:
0.7509 - val_loss: 0.2425 - val_acc: 0.7549
Epoch 11/20
57/57 [==============================] - 42s 741ms/step - loss: 0.2261 - acc:
0.7521 - val_loss: 0.2317 - val_acc: 0.7215
Epoch 12/20
57/57 [==============================] - 43s 748ms/step - loss: 0.2214 - acc:
0.7503 - val_loss: 0.2291 - val_acc: 0.7549
Epoch 13/20
57/57 [==============================] - 42s 739ms/step - loss: 0.2144 - acc:
0.7519 - val_loss: 0.2228 - val_acc: 0.7624
Epoch 14/20
57/57 [==============================] - 38s 663ms/step - loss: 0.2088 - acc:
0.7586 - val_loss: 0.2230 - val_acc: 0.7147
Epoch 15/20
57/57 [==============================] - 31s 544ms/step - loss: 0.2043 - acc:
0.7565 - val_loss: 0.2197 - val_acc: 0.7670
Epoch 16/20
57/57 [==============================] - 31s 536ms/step - loss: 0.1998 - acc:
0.7569 - val_loss: 0.2171 - val_acc: 0.7343
Epoch 17/20
57/57 [==============================] - 31s 543ms/step - loss: 0.1948 - acc:
0.7592 - val_loss: 0.2163 - val_acc: 0.7446
Epoch 18/20
57/57 [==============================] - 31s 543ms/step - loss: 0.1888 - acc:
0.7619 - val_loss: 0.2187 - val_acc: 0.7542
Epoch 19/20
57/57 [==============================] - 31s 540ms/step - loss: 0.1880 - acc:
0.7617 - val_loss: 0.2146 - val_acc: 0.7222
Epoch 20/20
57/57 [==============================] - 30s 532ms/step - loss: 0.1798 - acc:
0.7664 - val_loss: 0.2105 - val_acc: 0.7467
--- 800.39 seconds ---
Accuracy =  0.6298222800173385
Fraction Accuracy =  0.9116457159369958
Average Accuracy =  0.9116457159370034
```

0.8 case7

[40]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model

model = Sequential()
model.add(Embedding(vocab_size,
                    100,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False))
#model.add(SpatialDropout1D(0.3))
#model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3))
model.add(LSTM(128))
model.add(Dense(100, activation='relu'))
#model.add(Dropout(0.8))
#model.add(Dense(100, activation='relu'))
#model.add(Dropout(0.8))
model.add(Dense(6, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_10 (Embedding)     (None, 200, 100)          4194400
_________________________________________________________________
lstm_10 (LSTM)               (None, 128)               117248
_________________________________________________________________
dense_19 (Dense)             (None, 100)               12900
_________________________________________________________________
dense_20 (Dense)             (None, 6)                 606
=================================================================
Total params: 4,325,154
Trainable params: 130,754
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[41]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=300, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
```

```
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
38/38 [==============================] - 41s 1s/step - loss: 0.4914 - acc:
0.4012 - val_loss: 0.4322 - val_acc: 0.4194
Epoch 2/20
38/38 [==============================] - 40s 1s/step - loss: 0.4229 - acc:
0.4512 - val_loss: 0.4072 - val_acc: 0.5311
Epoch 3/20
38/38 [==============================] - 51s 1s/step - loss: 0.4023 - acc:
0.5184 - val_loss: 0.3829 - val_acc: 0.5553
Epoch 4/20
38/38 [==============================] - 50s 1s/step - loss: 0.3614 - acc:
0.6295 - val_loss: 0.3443 - val_acc: 0.6446
Epoch 5/20
38/38 [==============================] - 49s 1s/step - loss: 0.3252 - acc:
0.6720 - val_loss: 0.3242 - val_acc: 0.6770
Epoch 6/20
38/38 [==============================] - 50s 1s/step - loss: 0.3085 - acc:
0.7010 - val_loss: 0.3444 - val_acc: 0.6674
Epoch 7/20
38/38 [==============================] - 44s 1s/step - loss: 0.3168 - acc:
0.6861 - val_loss: 0.2961 - val_acc: 0.7108
Epoch 8/20
38/38 [==============================] - 37s 963ms/step - loss: 0.2888 - acc:
0.7267 - val_loss: 0.2906 - val_acc: 0.7293
Epoch 9/20
38/38 [==============================] - 35s 918ms/step - loss: 0.2895 - acc:
0.7207 - val_loss: 0.2881 - val_acc: 0.7325
Epoch 10/20
38/38 [==============================] - 35s 923ms/step - loss: 0.2779 - acc:
0.7356 - val_loss: 0.2759 - val_acc: 0.7335
Epoch 11/20
38/38 [==============================] - 35s 934ms/step - loss: 0.2707 - acc:
0.7383 - val_loss: 0.2673 - val_acc: 0.7385
Epoch 12/20
```

```
38/38 [==============================] - 35s 923ms/step - loss: 0.2631 - acc:
0.7400 - val_loss: 0.2628 - val_acc: 0.7335
Epoch 13/20
38/38 [==============================] - 35s 929ms/step - loss: 0.2575 - acc:
0.7391 - val_loss: 0.2550 - val_acc: 0.7264
Epoch 14/20
38/38 [==============================] - 35s 925ms/step - loss: 0.2627 - acc:
0.7256 - val_loss: 0.2646 - val_acc: 0.7375
Epoch 15/20
38/38 [==============================] - 35s 919ms/step - loss: 0.2544 - acc:
0.7270 - val_loss: 0.2545 - val_acc: 0.7293
Epoch 16/20
38/38 [==============================] - 35s 932ms/step - loss: 0.2466 - acc:
0.7307 - val_loss: 0.2500 - val_acc: 0.7446
Epoch 17/20
38/38 [==============================] - 36s 935ms/step - loss: 0.2418 - acc:
0.7307 - val_loss: 0.2438 - val_acc: 0.7506
Epoch 18/20
38/38 [==============================] - 35s 927ms/step - loss: 0.2341 - acc:
0.7394 - val_loss: 0.2356 - val_acc: 0.7115
Epoch 19/20
38/38 [==============================] - 35s 925ms/step - loss: 0.2285 - acc:
0.7372 - val_loss: 0.2324 - val_acc: 0.7271
Epoch 20/20
38/38 [==============================] - 35s 929ms/step - loss: 0.2233 - acc:
0.7457 - val_loss: 0.2332 - val_acc: 0.7535
--- 826.41 seconds ---
Accuracy =  0.6007802340702211
Fraction Accuracy =  0.9014352453884279
Average Accuracy =  0.9014352453884312
```

[ ]:

0.9 case9

[44]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout

model = Sequential()
model.add(Embedding(vocab_size,
                    100,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False))
```

```
#model.add(SpatialDropout1D(0.3))
#model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3))
#model.add(LSTM(128))
model.add(LSTM(128, dropout=0.3))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.3))
#model.add(Dense(100, activation='relu'))
#model.add(Dropout(0.8))
model.add(Dense(6, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_13 (Embedding)     (None, 200, 100)          4194400
_________________________________________________________________
lstm_13 (LSTM)               (None, 128)               117248
_________________________________________________________________
dense_22 (Dense)             (None, 100)               12900
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_23 (Dense)             (None, 6)                 606
=================================================================
Total params: 4,325,154
Trainable params: 130,754
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[45]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
```

```
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
88/88 [==============================] - 36s 414ms/step - loss: 0.4711 - acc:
0.3668 - val_loss: 0.4213 - val_acc: 0.4596
Epoch 2/20
88/88 [==============================] - 35s 400ms/step - loss: 0.4092 - acc:
0.5071 - val_loss: 0.3558 - val_acc: 0.6467
Epoch 3/20
88/88 [==============================] - 36s 410ms/step - loss: 0.3483 - acc:
0.6516 - val_loss: 0.2947 - val_acc: 0.7229
Epoch 4/20
88/88 [==============================] - 36s 408ms/step - loss: 0.2925 - acc:
0.7228 - val_loss: 0.2713 - val_acc: 0.7115
Epoch 5/20
88/88 [==============================] - 35s 402ms/step - loss: 0.2733 - acc:
0.7352 - val_loss: 0.2573 - val_acc: 0.7392
Epoch 6/20
88/88 [==============================] - 35s 402ms/step - loss: 0.2645 - acc:
0.7312 - val_loss: 0.2546 - val_acc: 0.7236
Epoch 7/20
88/88 [==============================] - 35s 402ms/step - loss: 0.2585 - acc:
0.7291 - val_loss: 0.2479 - val_acc: 0.6926
Epoch 8/20
88/88 [==============================] - 36s 407ms/step - loss: 0.2491 - acc:
0.7345 - val_loss: 0.2416 - val_acc: 0.7503
Epoch 9/20
88/88 [==============================] - 35s 401ms/step - loss: 0.2399 - acc:
0.7357 - val_loss: 0.2289 - val_acc: 0.7410
Epoch 10/20
88/88 [==============================] - 35s 403ms/step - loss: 0.2347 - acc:
0.7347 - val_loss: 0.2266 - val_acc: 0.7318
Epoch 11/20
88/88 [==============================] - 36s 405ms/step - loss: 0.2284 - acc:
0.7327 - val_loss: 0.2231 - val_acc: 0.7161
Epoch 12/20
88/88 [==============================] - 35s 401ms/step - loss: 0.2255 - acc:
0.7314 - val_loss: 0.2228 - val_acc: 0.7037
Epoch 13/20
88/88 [==============================] - 35s 403ms/step - loss: 0.2192 - acc:
0.7438 - val_loss: 0.2145 - val_acc: 0.7456
Epoch 14/20
88/88 [==============================] - 35s 400ms/step - loss: 0.2127 - acc:
0.7404 - val_loss: 0.2122 - val_acc: 0.7595
Epoch 15/20
```

```
88/88 [==============================] - 35s 403ms/step - loss: 0.2064 - acc:
0.7496 - val_loss: 0.2126 - val_acc: 0.7179
Epoch 16/20
88/88 [==============================] - 35s 402ms/step - loss: 0.2027 - acc:
0.7473 - val_loss: 0.2061 - val_acc: 0.7542
Epoch 17/20
88/88 [==============================] - 35s 401ms/step - loss: 0.2000 - acc:
0.7423 - val_loss: 0.2121 - val_acc: 0.7599
Epoch 18/20
88/88 [==============================] - 35s 401ms/step - loss: 0.1987 - acc:
0.7493 - val_loss: 0.2051 - val_acc: 0.7456
Epoch 19/20
88/88 [==============================] - 35s 401ms/step - loss: 0.1929 - acc:
0.7503 - val_loss: 0.2048 - val_acc: 0.7350
Epoch 20/20
88/88 [==============================] - 35s 403ms/step - loss: 0.1889 - acc:
0.7546 - val_loss: 0.2040 - val_acc: 0.7421
--- 735.97 seconds ---
Accuracy =  0.6366132061840775
Fraction Accuracy =  0.9154023984973164
Average Accuracy =  0.915402398497327
```

0.10 case10

[46]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout

model = Sequential()
model.add(Embedding(vocab_size,
                    100,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False))
#model.add(SpatialDropout1D(0.3))
#model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3))
#model.add(LSTM(128))
model.add(LSTM(128, dropout=0.5))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
#model.add(Dense(100, activation='relu'))
#model.add(Dropout(0.8))
model.add(Dense(6, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_14 (Embedding)     (None, 200, 100)          4194400
_________________________________________________________________
lstm_14 (LSTM)               (None, 128)               117248
_________________________________________________________________
dense_24 (Dense)             (None, 100)               12900
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_25 (Dense)             (None, 6)                 606
=================================================================
Total params: 4,325,154
Trainable params: 130,754
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[47]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
88/88 [==============================] - 42s 483ms/step - loss: 0.4867 - acc:
0.3451 - val_loss: 0.4265 - val_acc: 0.4216
Epoch 2/20
88/88 [==============================] - 50s 571ms/step - loss: 0.4285 - acc:
0.4560 - val_loss: 0.3973 - val_acc: 0.5379
Epoch 3/20
88/88 [==============================] - 47s 532ms/step - loss: 0.4033 - acc:
0.5480 - val_loss: 0.3885 - val_acc: 0.5219
```

```
Epoch 4/20
88/88 [==============================] - 47s 532ms/step - loss: 0.3946 - acc:
0.5271 - val_loss: 0.3751 - val_acc: 0.5503
Epoch 5/20
88/88 [==============================] - 46s 528ms/step - loss: 0.3623 - acc:
0.6242 - val_loss: 0.3146 - val_acc: 0.6912
Epoch 6/20
88/88 [==============================] - 46s 526ms/step - loss: 0.3242 - acc:
0.6988 - val_loss: 0.2911 - val_acc: 0.7204
Epoch 7/20
88/88 [==============================] - 39s 442ms/step - loss: 0.2988 - acc:
0.7228 - val_loss: 0.2727 - val_acc: 0.7332
Epoch 8/20
88/88 [==============================] - 37s 417ms/step - loss: 0.2845 - acc:
0.7307 - val_loss: 0.2664 - val_acc: 0.7545
Epoch 9/20
88/88 [==============================] - 37s 416ms/step - loss: 0.2740 - acc:
0.7419 - val_loss: 0.2674 - val_acc: 0.7311
Epoch 10/20
88/88 [==============================] - 36s 408ms/step - loss: 0.2732 - acc:
0.7378 - val_loss: 0.2532 - val_acc: 0.7353
Epoch 11/20
88/88 [==============================] - 35s 401ms/step - loss: 0.2666 - acc:
0.7361 - val_loss: 0.2455 - val_acc: 0.7510
Epoch 12/20
88/88 [==============================] - 36s 404ms/step - loss: 0.2594 - acc:
0.7408 - val_loss: 0.2456 - val_acc: 0.7097
Epoch 13/20
88/88 [==============================] - 36s 405ms/step - loss: 0.2577 - acc:
0.7362 - val_loss: 0.2484 - val_acc: 0.7453
Epoch 14/20
88/88 [==============================] - 36s 407ms/step - loss: 0.2535 - acc:
0.7391 - val_loss: 0.2347 - val_acc: 0.7464
Epoch 15/20
88/88 [==============================] - 36s 406ms/step - loss: 0.2493 - acc:
0.7433 - val_loss: 0.2367 - val_acc: 0.7432
Epoch 16/20
88/88 [==============================] - 36s 407ms/step - loss: 0.2477 - acc:
0.7360 - val_loss: 0.2301 - val_acc: 0.7400
Epoch 17/20
88/88 [==============================] - 36s 406ms/step - loss: 0.2437 - acc:
0.7431 - val_loss: 0.2280 - val_acc: 0.7627
Epoch 18/20
88/88 [==============================] - 35s 403ms/step - loss: 0.2424 - acc:
0.7399 - val_loss: 0.2234 - val_acc: 0.7378
Epoch 19/20
88/88 [==============================] - 36s 406ms/step - loss: 0.2367 - acc:
0.7437 - val_loss: 0.2254 - val_acc: 0.7410
```

```
Epoch 20/20
88/88 [==============================] - 36s 405ms/step - loss: 0.2379 - acc:
0.7419 - val_loss: 0.2228 - val_acc: 0.7681
--- 813.05 seconds ---
Accuracy =  0.615662476520734
Fraction Accuracy =  0.9072629196166184
Average Accuracy =  0.9072629196166258
```

0.11 case11

[48]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout

model = Sequential()
model.add(Embedding(vocab_size,
                    100,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False))
#model.add(SpatialDropout1D(0.3))
#model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3))
#model.add(LSTM(128))
model.add(LSTM(128, dropout=0.3))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.8))
#model.add(Dense(100, activation='relu'))
#model.add(Dropout(0.8))
model.add(Dense(6, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_15 (Embedding)     (None, 200, 100)          4194400
_________________________________________________________________
lstm_15 (LSTM)               (None, 128)               117248
_________________________________________________________________
dense_26 (Dense)             (None, 100)               12900
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_27 (Dense)             (None, 6)                 606
=================================================================
```

```
Total params: 4,325,154
Trainable params: 130,754
Non-trainable params: 4,194,400

_________________________________________________________________
None
```

[49]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
88/88 [==============================] - 43s 486ms/step - loss: 0.5152 - acc:
0.3164 - val_loss: 0.4333 - val_acc: 0.4244
Epoch 2/20
88/88 [==============================] - 39s 444ms/step - loss: 0.4483 - acc:
0.4141 - val_loss: 0.3942 - val_acc: 0.5247
Epoch 3/20
88/88 [==============================] - 37s 423ms/step - loss: 0.4295 - acc:
0.4653 - val_loss: 0.4155 - val_acc: 0.5116
Epoch 4/20
88/88 [==============================] - 39s 446ms/step - loss: 0.4093 - acc:
0.5298 - val_loss: 0.3441 - val_acc: 0.6489
Epoch 5/20
88/88 [==============================] - 44s 504ms/step - loss: 0.3710 - acc:
0.6049 - val_loss: 0.3323 - val_acc: 0.6311
Epoch 6/20
88/88 [==============================] - 55s 626ms/step - loss: 0.3444 - acc:
0.6816 - val_loss: 0.2967 - val_acc: 0.7307
Epoch 7/20
88/88 [==============================] - 53s 604ms/step - loss: 0.3280 - acc:
0.7044 - val_loss: 0.2926 - val_acc: 0.7314
Epoch 8/20
88/88 [==============================] - 50s 569ms/step - loss: 0.3312 - acc:
```

```
0.6903 - val_loss: 0.2834 - val_acc: 0.7350
Epoch 9/20
88/88 [==============================] - 47s 533ms/step - loss: 0.3134 - acc:
0.7221 - val_loss: 0.2827 - val_acc: 0.7396
Epoch 10/20
88/88 [==============================] - 44s 500ms/step - loss: 0.3095 - acc:
0.7168 - val_loss: 0.3049 - val_acc: 0.6834
Epoch 11/20
88/88 [==============================] - 36s 408ms/step - loss: 0.3091 - acc:
0.7214 - val_loss: 0.2848 - val_acc: 0.7243
Epoch 12/20
88/88 [==============================] - 43s 484ms/step - loss: 0.3040 - acc:
0.7238 - val_loss: 0.2647 - val_acc: 0.7488
Epoch 13/20
88/88 [==============================] - 40s 454ms/step - loss: 0.2906 - acc:
0.7347 - val_loss: 0.2631 - val_acc: 0.7432
Epoch 14/20
88/88 [==============================] - 51s 583ms/step - loss: 0.2888 - acc:
0.7380 - val_loss: 0.2555 - val_acc: 0.7492
Epoch 15/20
88/88 [==============================] - 53s 607ms/step - loss: 0.2785 - acc:
0.7459 - val_loss: 0.2558 - val_acc: 0.7478
Epoch 16/20
88/88 [==============================] - 54s 619ms/step - loss: 0.2792 - acc:
0.7467 - val_loss: 0.2540 - val_acc: 0.7549
Epoch 17/20
88/88 [==============================] - 51s 583ms/step - loss: 0.2729 - acc:
0.7472 - val_loss: 0.2493 - val_acc: 0.7595
Epoch 18/20
88/88 [==============================] - 53s 606ms/step - loss: 0.2782 - acc:
0.7418 - val_loss: 0.2508 - val_acc: 0.7528
Epoch 19/20
88/88 [==============================] - 53s 608ms/step - loss: 0.2707 - acc:
0.7504 - val_loss: 0.2442 - val_acc: 0.7602
Epoch 20/20
88/88 [==============================] - 42s 482ms/step - loss: 0.2657 - acc:
0.7475 - val_loss: 0.2466 - val_acc: 0.7584
--- 960.57 seconds ---
Accuracy =  0.5679815055627799
Fraction Accuracy =  0.8885035881134699
Average Accuracy =  0.888503588113471
```

0.12 case12

[12]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, LSTM
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout

model = Sequential()
model.add(Embedding(vocab_size,
                    100,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False))
#model.add(SpatialDropout1D(0.3))
#model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3))
#model.add(LSTM(128))
model.add(LSTM(500, dropout=0.3))
model.add(Dense(300, activation='relu'))
model.add(Dropout(0.3))
#model.add(Dense(100, activation='relu'))
#model.add(Dropout(0.8))
model.add(Dense(6, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 200, 100)          4194400
_________________________________________________________________
lstm (LSTM)                  (None, 500)               1202000
_________________________________________________________________
dense (Dense)                (None, 300)               150300
_________________________________________________________________
dropout (Dropout)            (None, 300)               0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 1806
=================================================================
Total params: 5,548,506
Trainable params: 1,354,106
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[13]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
```

```
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
88/88 [==============================] - 288s 3s/step - loss: 0.4410 - acc:
0.4147 - val_loss: 0.4022 - val_acc: 0.5165
Epoch 2/20
88/88 [==============================] - 358s 4s/step - loss: 0.3839 - acc:
0.5696 - val_loss: 0.3890 - val_acc: 0.5735
Epoch 3/20
88/88 [==============================] - 251s 3s/step - loss: 0.3467 - acc:
0.6555 - val_loss: 0.2951 - val_acc: 0.7118
Epoch 4/20
88/88 [==============================] - 250s 3s/step - loss: 0.3433 - acc:
0.6202 - val_loss: 0.3876 - val_acc: 0.4988
Epoch 5/20
88/88 [==============================] - 249s 3s/step - loss: 0.3239 - acc:
0.6688 - val_loss: 0.2673 - val_acc: 0.7414
Epoch 6/20
88/88 [==============================] - 246s 3s/step - loss: 0.2695 - acc:
0.7330 - val_loss: 0.2607 - val_acc: 0.7485
Epoch 7/20
88/88 [==============================] - 250s 3s/step - loss: 0.2583 - acc:
0.7323 - val_loss: 0.2466 - val_acc: 0.7545
Epoch 8/20
88/88 [==============================] - 250s 3s/step - loss: 0.2472 - acc:
0.7289 - val_loss: 0.2399 - val_acc: 0.7460
Epoch 9/20
88/88 [==============================] - 247s 3s/step - loss: 0.2383 - acc:
0.7335 - val_loss: 0.2374 - val_acc: 0.7375
Epoch 10/20
88/88 [==============================] - 246s 3s/step - loss: 0.2311 - acc:
0.7379 - val_loss: 0.2291 - val_acc: 0.7645
Epoch 11/20
88/88 [==============================] - 247s 3s/step - loss: 0.2212 - acc:
0.7431 - val_loss: 0.2216 - val_acc: 0.7318
Epoch 12/20
```

```
88/88 [==============================] - 245s 3s/step - loss: 0.2178 - acc:
0.7463 - val_loss: 0.2139 - val_acc: 0.7456
Epoch 13/20
88/88 [==============================] - 234s 3s/step - loss: 0.2106 - acc:
0.7428 - val_loss: 0.2086 - val_acc: 0.7524
Epoch 14/20
88/88 [==============================] - 234s 3s/step - loss: 0.2047 - acc:
0.7499 - val_loss: 0.2088 - val_acc: 0.7385
Epoch 15/20
88/88 [==============================] - 234s 3s/step - loss: 0.1999 - acc:
0.7503 - val_loss: 0.2134 - val_acc: 0.7577
Epoch 16/20
88/88 [==============================] - 233s 3s/step - loss: 0.1988 - acc:
0.7506 - val_loss: 0.2104 - val_acc: 0.7172
Epoch 17/20
88/88 [==============================] - 233s 3s/step - loss: 0.1918 - acc:
0.7535 - val_loss: 0.2021 - val_acc: 0.7346
Epoch 18/20
88/88 [==============================] - 234s 3s/step - loss: 0.1908 - acc:
0.7527 - val_loss: 0.2061 - val_acc: 0.7503
Epoch 19/20
88/88 [==============================] - 233s 3s/step - loss: 0.1860 - acc:
0.7548 - val_loss: 0.2071 - val_acc: 0.7595
Epoch 20/20
88/88 [==============================] - 234s 3s/step - loss: 0.1806 - acc:
0.7581 - val_loss: 0.2054 - val_acc: 0.7645
--- 5127.17 seconds ---
Accuracy =  0.6331454992053172


 ↪----------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
 ↪last)

       <ipython-input-13-d9c357e4b79e> in <module>
        11 # calculate accuracy
        12 print("Accuracy = ", accuracy_score(y_test, predicted))
   ---> 13 print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
        14 accuracy_label = []
        15 for i in list(range(0,6)):


       NameError: name 'accuracy_fraction' is not defined
```

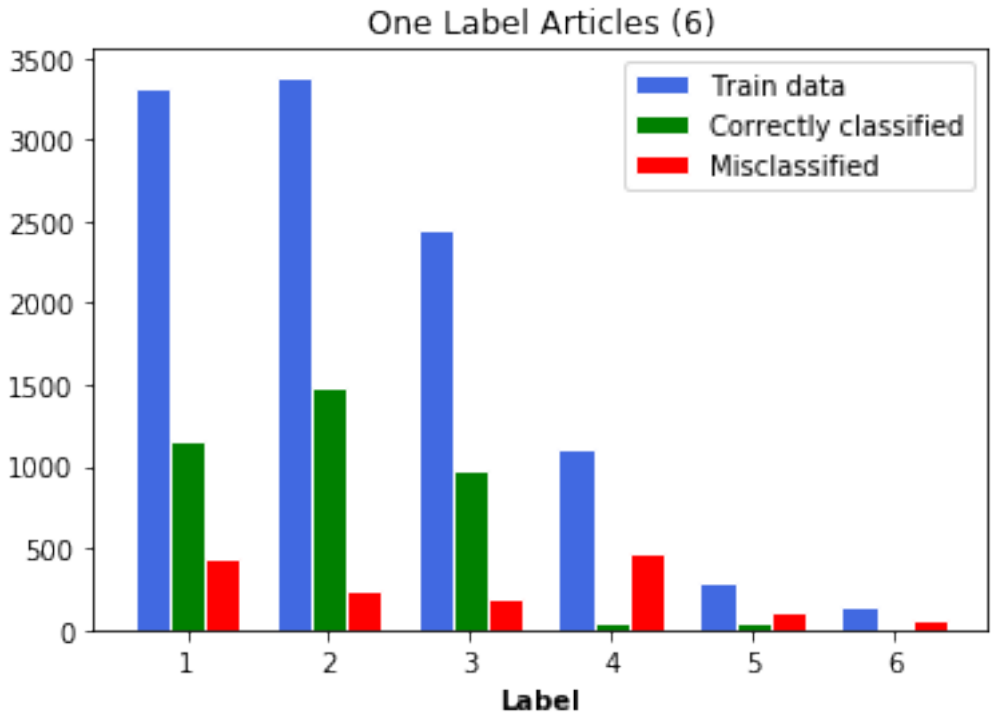
[14]:
```
misclassification = (y_test == predicted).all(axis=1)
i, = np.where(misclassification == False)
j, = np.where(misclassification == True)
mis_y_test = y_test[i]
correct_y_test = y_test[j]
print(len(mis_y_test))
print(len(correct_y_test))
print(len(y_test))
```

```
2539
4382
6921
```

[17]:
```
#mis articles that only belong to 1 label.
x = np.array([[1, 0, 0, 0, 0, 0],
              [0, 1, 0, 0, 0, 0],
              [0, 0, 1, 0, 0, 0],
              [0, 0, 0, 1, 0, 0],
              [0, 0, 0, 0, 1, 0],
              [0, 0, 0, 0, 0, 1]])
c = x.shape[0]
mis_num = []
correct_num = []
train_num = []

for i in range(0,c):
    each_combine = np.repeat(([x[i,:]]), len(mis_y_test), axis=0)
    paired_mis_num = accuracy_score(mis_y_test, each_combine, False)
    mis_num.append(paired_mis_num)

    each_combine = np.repeat(([x[i,:]]), len(correct_y_test), axis=0)
    paired_correct_num = accuracy_score(correct_y_test, each_combine, False)
    correct_num.append(paired_correct_num)

    each_combine = np.repeat(([x[i,:]]), len(y_train), axis=0)
    paired_train_num = accuracy_score(y_train, each_combine, False)
    train_num.append(paired_train_num)

print(mis_num)
print(correct_num)
print(train_num)

#plot
import matplotlib.pyplot as plt
# set width of bar
barWidth = 0.25
# Set position of bar on X axis
```

```
r1 = np.arange(len(correct_num))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

# Make the plot
plt.bar(r1, train_num, color='royalblue', width=barWidth, edgecolor='white',␣
 ↪label='Train data')
plt.bar(r2, correct_num, color='green', width=barWidth, edgecolor='white',␣
 ↪label='Correctly classified')
plt.bar(r3, mis_num, color='red', width=barWidth, edgecolor='white',␣
 ↪label='Misclassified')

# Add xticks on the middle of the group bars
plt.title('One Label Articles (6)')
plt.xlabel('Label', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(correct_num))], ["1" ,"2" , "3",␣
 ↪"4" , "5" , "6"])

# Create legend & Show graphic
plt.legend()
plt.savefig('NN_1.pdf',bbox_inches='tight')
plt.show()
```

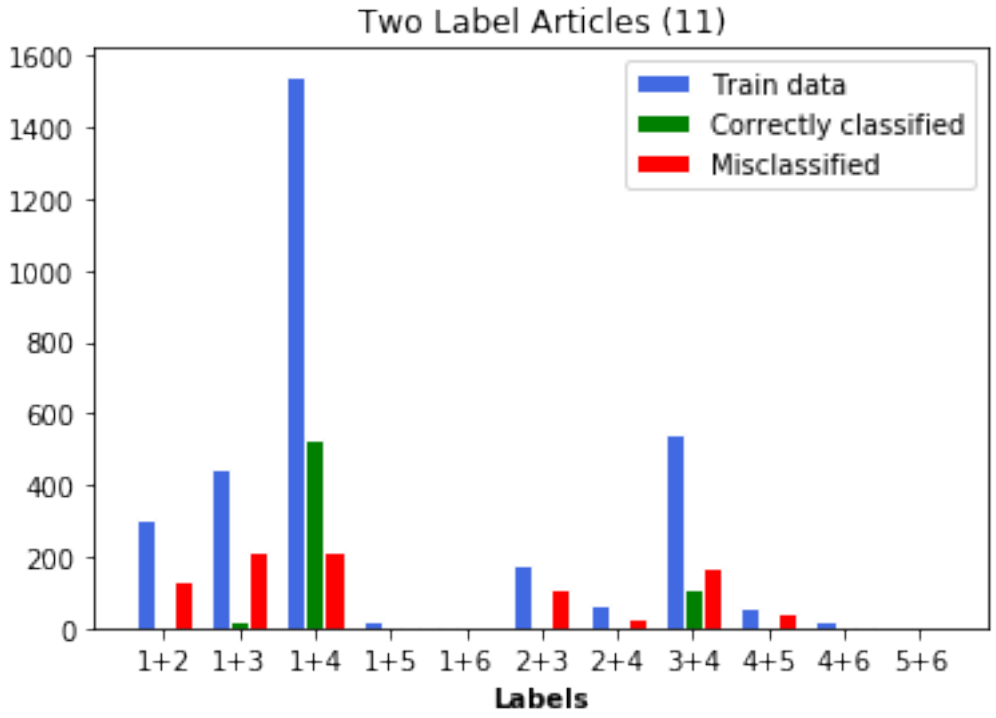
```
[434, 244, 187, 467, 116, 64]
[1156, 1490, 975, 53, 39, 0]
[3320, 3386, 2448, 1116, 288, 145]
```



[19]:
```
#mis articles that only belong to 2 label.
x = np.array([[1, 1, 0, 0, 0, 0],
              [1, 0, 1, 0, 0, 0],
              [1, 0, 0, 1, 0, 0],
              [1, 0, 0, 0, 1, 0],
              [1, 0, 0, 0, 0, 1],
              [0, 1, 1, 0, 0, 0],
              [0, 1, 0, 1, 0, 0],
              [0, 0, 1, 1, 0, 0],
              [0, 0, 0, 1, 1, 0],
              [0, 0, 0, 1, 0, 1],
              [0, 0, 0, 0, 1, 1]])

c = x.shape[0]
mis_num = []
correct_num = []
train_num = []

for i in range(0,c):
    each_combine = np.repeat(([x[i,:]]), len(mis_y_test), axis=0)
    paired_mis_num = accuracy_score(mis_y_test, each_combine, False)
    mis_num.append(paired_mis_num)

    each_combine = np.repeat(([x[i,:]]), len(correct_y_test), axis=0)
    paired_correct_num = accuracy_score(correct_y_test, each_combine, False)
    correct_num.append(paired_correct_num)

    each_combine = np.repeat(([x[i,:]]), len(y_train), axis=0)
    paired_train_num = accuracy_score(y_train, each_combine, False)
    train_num.append(paired_train_num)

print(mis_num)
print(correct_num)
print(train_num)

#plot
import matplotlib.pyplot as plt
# set width of bar
barWidth = 0.25
# Set position of bar on X axis
r1 = np.arange(len(correct_num))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

# Make the plot
```

```
plt.bar(r1, train_num, color='royalblue', width=barWidth, edgecolor='white',␣
 ↪label='Train data')
plt.bar(r2, correct_num, color='green', width=barWidth, edgecolor='white',␣
 ↪label='Correctly classified')
plt.bar(r3, mis_num, color='red', width=barWidth, edgecolor='white',␣
 ↪label='Misclassified')

# Add xticks on the middle of the group bars
target_labels = ["1+2" ,"1+3" , "1+4", "1+5" , "1+6" , "2+3", "2+4", "3+4",␣
 ↪"4+5",
                  "4+6", "5+6"]
plt.title('Two Label Articles (11)')
plt.xlabel('Labels', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(correct_num))], target_labels)

# Create legend & Show graphic
plt.legend()
plt.savefig('NN_2.pdf',bbox_inches='tight')
plt.show()
```

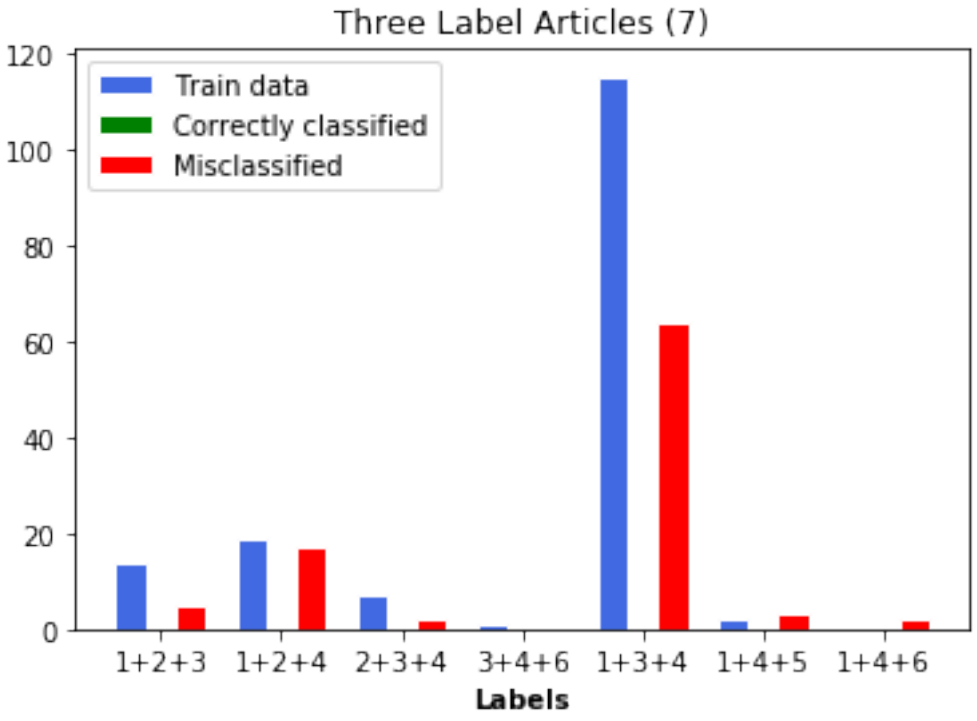
```
[133, 215, 214, 8, 0, 110, 31, 172, 45, 4, 2]
[1, 20, 527, 0, 0, 9, 0, 112, 0, 0, 0]
[303, 447, 1544, 22, 9, 174, 68, 541, 60, 20, 2]
```



[20]:
```
#articles that only belong to 3 label.
x = np.array([[1, 1, 1, 0, 0, 0],
              [1, 1, 0, 1, 0, 0],
              [0, 1, 1, 1, 0, 0],
              [0, 0, 1, 1, 0, 1],
              [1, 0, 1, 1, 0, 0],
              [1, 0, 0, 1, 1, 0],
              [1, 0, 0, 1, 0, 1]])
c = x.shape[0]
mis_num = []
correct_num = []
train_num = []

for i in range(0,c):
    each_combine = np.repeat(([x[i,:]]), len(mis_y_test), axis=0)
    paired_mis_num = accuracy_score(mis_y_test, each_combine, False)
    mis_num.append(paired_mis_num)

    each_combine = np.repeat(([x[i,:]]), len(correct_y_test), axis=0)
    paired_correct_num = accuracy_score(correct_y_test, each_combine, False)
    correct_num.append(paired_correct_num)

    each_combine = np.repeat(([x[i,:]]), len(y_train), axis=0)
    paired_train_num = accuracy_score(y_train, each_combine, False)
    train_num.append(paired_train_num)

print(mis_num)
print(correct_num)
print(train_num)

#plot
import matplotlib.pyplot as plt
# set width of bar
barWidth = 0.25
# Set position of bar on X axis
r1 = np.arange(len(correct_num))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]

# Make the plot
plt.bar(r1, train_num, color='royalblue', width=barWidth, edgecolor='white',␣
 ↪label='Train data')
plt.bar(r2, correct_num, color='green', width=barWidth, edgecolor='white',␣
 ↪label='Correctly classified')
plt.bar(r3, mis_num, color='red', width=barWidth, edgecolor='white',␣
 ↪label='Misclassified')
```

```
# Add xticks on the middle of the group bars
target_labels = ["1+2+3" ,"1+2+4" , "2+3+4", "3+4+6" , "1+3+4" , "1+4+5",␣
↪"1+4+6"]
plt.title('Three Label Articles (7)')
plt.xlabel('Labels', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(correct_num))], target_labels)

# Create legend & Show graphic
plt.legend()
plt.savefig('NN_3.pdf',bbox_inches='tight')
plt.show()
```

```
[5, 17, 2, 0, 64, 3, 2]
[0, 0, 0, 0, 0, 0, 0]
[14, 19, 7, 1, 115, 2, 0]
```



[ ]:

0.13 case13

[52]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, LSTM
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout

model = Sequential()
model.add(Embedding(vocab_size,
                    100,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False))
#model.add(SpatialDropout1D(0.3))
#model.add(LSTM(128, dropout=0.3, recurrent_dropout=0.3))
#model.add(LSTM(128))
model.add(LSTM(500, dropout=0.3))
model.add(Dense(300, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(300, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(6, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_17 (Embedding)     (None, 200, 100)          4194400
_________________________________________________________________
lstm_17 (LSTM)               (None, 500)               1202000
_________________________________________________________________
dense_30 (Dense)             (None, 300)               150300
_________________________________________________________________
dropout_4 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_31 (Dense)             (None, 300)               90300
_________________________________________________________________
dropout_5 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_32 (Dense)             (None, 6)                 1806
=================================================================
Total params: 5,638,806
Trainable params: 1,444,406
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[53]:
```
import time
start_time = time.time()
```

```
model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
88/88 [==============================] - 266s 3s/step - loss: 0.4462 - acc:
0.4002 - val_loss: 0.3951 - val_acc: 0.5432
Epoch 2/20
88/88 [==============================] - 286s 3s/step - loss: 0.4138 - acc:
0.4831 - val_loss: 0.3938 - val_acc: 0.5400
Epoch 3/20
88/88 [==============================] - 330s 4s/step - loss: 0.3730 - acc:
0.5926 - val_loss: 0.3022 - val_acc: 0.7254
Epoch 4/20
88/88 [==============================] - 297s 3s/step - loss: 0.3006 - acc:
0.7192 - val_loss: 0.2820 - val_acc: 0.7207
Epoch 5/20
88/88 [==============================] - 312s 4s/step - loss: 0.2771 - acc:
0.7343 - val_loss: 0.2609 - val_acc: 0.7474
Epoch 6/20
88/88 [==============================] - 298s 3s/step - loss: 0.2630 - acc:
0.7399 - val_loss: 0.2555 - val_acc: 0.7037
Epoch 7/20
88/88 [==============================] - 293s 3s/step - loss: 0.2554 - acc:
0.7411 - val_loss: 0.2607 - val_acc: 0.7247
Epoch 8/20
88/88 [==============================] - 269s 3s/step - loss: 0.2521 - acc:
0.7376 - val_loss: 0.2460 - val_acc: 0.7367
Epoch 9/20
88/88 [==============================] - 269s 3s/step - loss: 0.2422 - acc:
0.7473 - val_loss: 0.2434 - val_acc: 0.7602
Epoch 10/20
88/88 [==============================] - 281s 3s/step - loss: 0.2330 - acc:
0.7431 - val_loss: 0.2356 - val_acc: 0.7432
```

```
Epoch 11/20
88/88 [==============================] - 282s 3s/step - loss: 0.2269 - acc:
0.7448 - val_loss: 0.2228 - val_acc: 0.7360
Epoch 12/20
88/88 [==============================] - 350s 4s/step - loss: 0.2209 - acc:
0.7461 - val_loss: 0.2187 - val_acc: 0.7474
Epoch 13/20
88/88 [==============================] - 339s 4s/step - loss: 0.2149 - acc:
0.7521 - val_loss: 0.2159 - val_acc: 0.7513
Epoch 14/20
88/88 [==============================] - 271s 3s/step - loss: 0.2094 - acc:
0.7481 - val_loss: 0.2101 - val_acc: 0.7609
Epoch 15/20
88/88 [==============================] - 264s 3s/step - loss: 0.2047 - acc:
0.7505 - val_loss: 0.2090 - val_acc: 0.7595
Epoch 16/20
88/88 [==============================] - 269s 3s/step - loss: 0.1978 - acc:
0.7533 - val_loss: 0.2069 - val_acc: 0.7150
Epoch 17/20
88/88 [==============================] - 268s 3s/step - loss: 0.1950 - acc:
0.7559 - val_loss: 0.2130 - val_acc: 0.7520
Epoch 18/20
88/88 [==============================] - 257s 3s/step - loss: 0.1905 - acc:
0.7577 - val_loss: 0.2137 - val_acc: 0.7282
Epoch 19/20
88/88 [==============================] - 274s 3s/step - loss: 0.1857 - acc:
0.7609 - val_loss: 0.2149 - val_acc: 0.7620
Epoch 20/20
88/88 [==============================] - 270s 3s/step - loss: 0.1831 - acc:
0.7657 - val_loss: 0.2051 - val_acc: 0.7467
--- 5896.75 seconds ---
Accuracy =  0.6408033521167461
Fraction Accuracy =  0.9148966912295825
Average Accuracy =  0.9148966912295912
```

0.14 case14

[54]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, LSTM
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout,SpatialDropout1D

model = Sequential()
model.add(Embedding(vocab_size,
                    100,
                    weights=[embedding_matrix],
```

```
                    input_length=maxlen,
                    trainable=False))
model.add(SpatialDropout1D(0.3))
model.add(LSTM(500, dropout=0.3))
model.add(Dense(300, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(6, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_18 (Embedding)     (None, 200, 100)          4194400
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 200, 100)          0
_________________________________________________________________
lstm_18 (LSTM)               (None, 500)               1202000
_________________________________________________________________
dense_33 (Dense)             (None, 300)               150300
_________________________________________________________________
dropout_6 (Dropout)          (None, 300)               0
_________________________________________________________________
dense_34 (Dense)             (None, 6)                 1806
=================================================================
Total params: 5,548,506
Trainable params: 1,354,106
Non-trainable params: 4,194,400
_________________________________________________________________
None
```

[55]:
```
import time
start_time = time.time()
model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1,␣
 ↪validation_split=0.2)
# make a prediction on the test set
predicted = model.predict(X_test)
print("--- %s seconds ---" % round(time.time() - start_time,2))

from sklearn.metrics import accuracy_score
# round probabilities to class labels
predicted = predicted.round()
# calculate accuracy
print("Accuracy = ", accuracy_score(y_test, predicted))
print("Fraction Accuracy = ", accuracy_fraction(y_test, predicted))
accuracy_label = []
```

```
for i in list(range(0,6)):
    accuracy_label.append(accuracy_score(y_test[:,i],predicted[:,i]))
print("Average Accuracy = ", sum(accuracy_label)/6)
```

```
Epoch 1/20
88/88 [==============================] - 280s 3s/step - loss: 0.4484 - acc:
0.3876 - val_loss: 0.4255 - val_acc: 0.4055
Epoch 2/20
88/88 [==============================] - 278s 3s/step - loss: 0.4147 - acc:
0.4811 - val_loss: 0.3942 - val_acc: 0.5251
Epoch 3/20
88/88 [==============================] - 294s 3s/step - loss: 0.3898 - acc:
0.5440 - val_loss: 0.3555 - val_acc: 0.6009
Epoch 4/20
88/88 [==============================] - 411s 5s/step - loss: 0.3245 - acc:
0.6818 - val_loss: 0.2978 - val_acc: 0.7197
Epoch 5/20
88/88 [==============================] - 283s 3s/step - loss: 0.2951 - acc:
0.7202 - val_loss: 0.2739 - val_acc: 0.7364
Epoch 6/20
88/88 [==============================] - 252s 3s/step - loss: 0.2780 - acc:
0.7274 - val_loss: 0.2617 - val_acc: 0.7464
Epoch 7/20
88/88 [==============================] - 282s 3s/step - loss: 0.2693 - acc:
0.7211 - val_loss: 0.2646 - val_acc: 0.7186
Epoch 8/20
88/88 [==============================] - 279s 3s/step - loss: 0.2611 - acc:
0.7109 - val_loss: 0.2398 - val_acc: 0.7229
Epoch 9/20
88/88 [==============================] - 280s 3s/step - loss: 0.2546 - acc:
0.7197 - val_loss: 0.2394 - val_acc: 0.7150
Epoch 10/20
88/88 [==============================] - 288s 3s/step - loss: 0.2482 - acc:
0.7191 - val_loss: 0.2333 - val_acc: 0.7492
Epoch 11/20
88/88 [==============================] - 270s 3s/step - loss: 0.2403 - acc:
0.7228 - val_loss: 0.2307 - val_acc: 0.6912
Epoch 12/20
88/88 [==============================] - 246s 3s/step - loss: 0.2367 - acc:
0.7263 - val_loss: 0.2222 - val_acc: 0.7435
Epoch 13/20
88/88 [==============================] - 247s 3s/step - loss: 0.2291 - acc:
0.7324 - val_loss: 0.2197 - val_acc: 0.7062
Epoch 14/20
88/88 [==============================] - 248s 3s/step - loss: 0.2273 - acc:
0.7282 - val_loss: 0.2176 - val_acc: 0.7656
Epoch 15/20
```

```
88/88 [==============================] - 247s 3s/step - loss: 0.2256 - acc:
0.7359 - val_loss: 0.2145 - val_acc: 0.7357
Epoch 16/20
88/88 [==============================] - 244s 3s/step - loss: 0.2217 - acc:
0.7287 - val_loss: 0.2079 - val_acc: 0.7417
Epoch 17/20
88/88 [==============================] - 238s 3s/step - loss: 0.2177 - acc:
0.7338 - val_loss: 0.2040 - val_acc: 0.7510
Epoch 18/20
88/88 [==============================] - 237s 3s/step - loss: 0.2148 - acc:
0.7396 - val_loss: 0.2080 - val_acc: 0.7400
Epoch 19/20
88/88 [==============================] - 238s 3s/step - loss: 0.2113 - acc:
0.7366 - val_loss: 0.2083 - val_acc: 0.7371
Epoch 20/20
88/88 [==============================] - 238s 3s/step - loss: 0.2075 - acc:
0.7403 - val_loss: 0.2018 - val_acc: 0.7567
--- 5528.76 seconds ---
Accuracy =  0.6422482300245629
Fraction Accuracy =  0.9155950488850279
Average Accuracy =  0.9155950488850358
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[21]:
```python
#Calculate test accuracy by fractional values
def accuracy_fraction(y_test, predicted):
    test_size = y_test.shape[0]
    frac_values = []
    for i in range(0,test_size):
        single_frac = accuracy_score(y_test[i,], predicted[i,])
        frac_values.append(single_frac)
    test_scores = sum(frac_values)/test_size
    return test_scores
```

[38]:
```python
score = model.evaluate(X_test, y_test, verbose=1)

print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
217/217 [==============================] - 10s 47ms/step - loss: 0.2776 - acc:
```

```
0.7451
Test Score: 0.2775523066520691
Test Accuracy: 0.745123565196991
```

[42]:
```
X_train.shape
```

[42]:
```
(14051, 1000)
```

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout,SpatialDropout1D

model = Sequential()
model.add(Embedding(vocab_size,
                    100,
                    weights=[embedding_matrix],
                    input_length=maxlen,
                    trainable=False))
model.add(SpatialDropout1D(0.3))
model.add(LSTM(500, dropout=0.3))
model.add(Dense(300, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(6, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```