neural_network

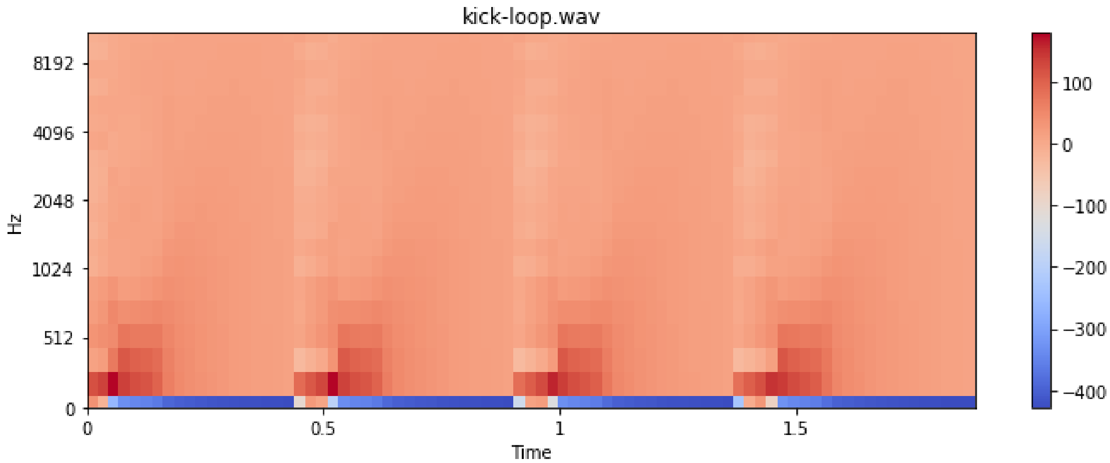
September 27, 2019

```
In [1]: import librosa
        import librosa.feature
        import librosa.display
        import glob
        import numpy as np
        import matplotlib.pyplot as plt
        from keras.models import Sequential
        from keras.layers import Dense, Activation
        from keras.utils.np_utils import to_categorical

        #Using TensoFlow backend.
```

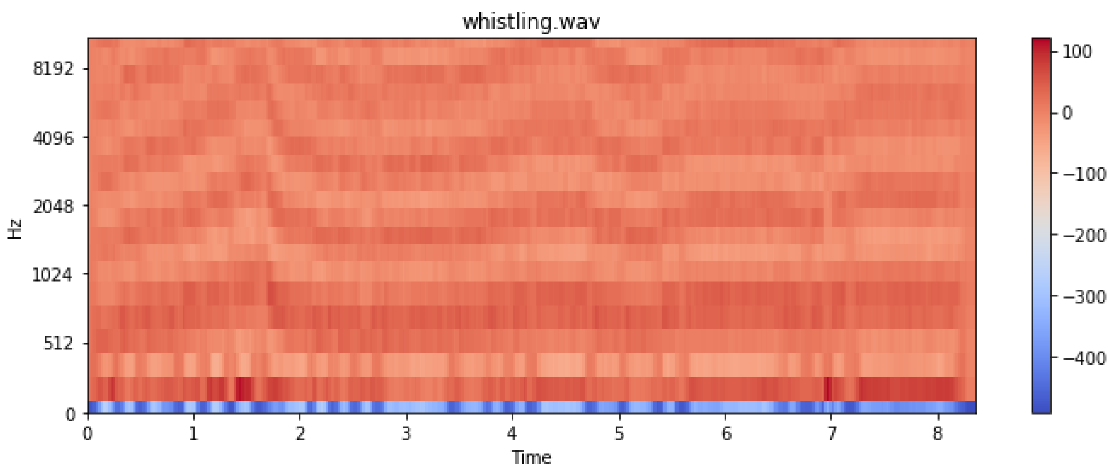
```
Using TensorFlow backend.
d:\dev\python\python36\lib\site-packages\tensorflow\python\framework\dtypes.py:523: FutureWarn
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
d:\dev\python\python36\lib\site-packages\tensorflow\python\framework\dtypes.py:524: FutureWarn
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
d:\dev\python\python36\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarn
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
d:\dev\python\python36\lib\site-packages\tensorflow\python\framework\dtypes.py:526: FutureWarn
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
d:\dev\python\python36\lib\site-packages\tensorflow\python\framework\dtypes.py:527: FutureWarn
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
d:\dev\python\python36\lib\site-packages\tensorflow\python\framework\dtypes.py:532: FutureWarn
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

```
In [5]: def display_mfcc(song):
            y, _ = librosa.load(song)
            mfcc = librosa.feature.mfcc(y)

            plt.figure(figsize=(10,4))
            librosa.display.specshow(mfcc, x_axis='time', y_axis='mel')
            plt.colorbar()
            plt.title(song)
            plt.tight_layout()
            plt.show()
```

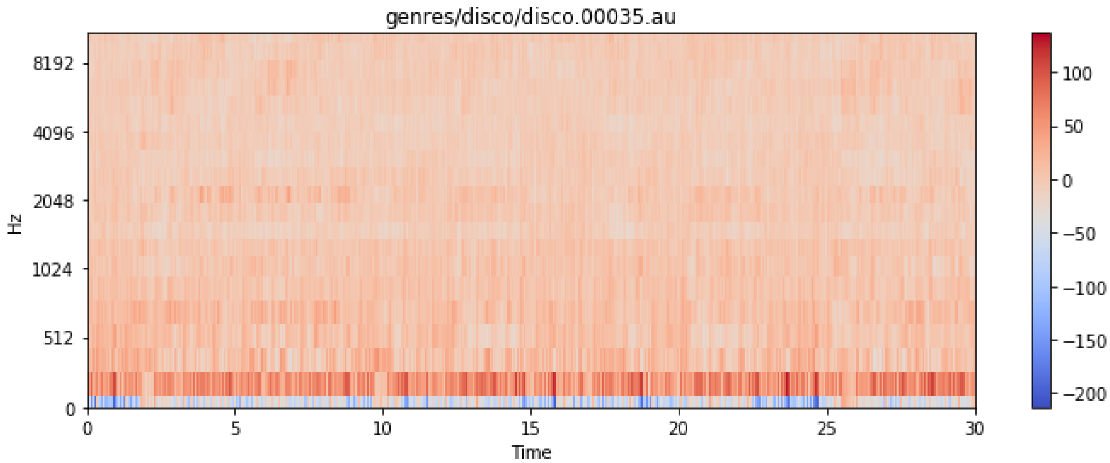
```
In [9]: display_mfcc('kick-loop.wav')
```



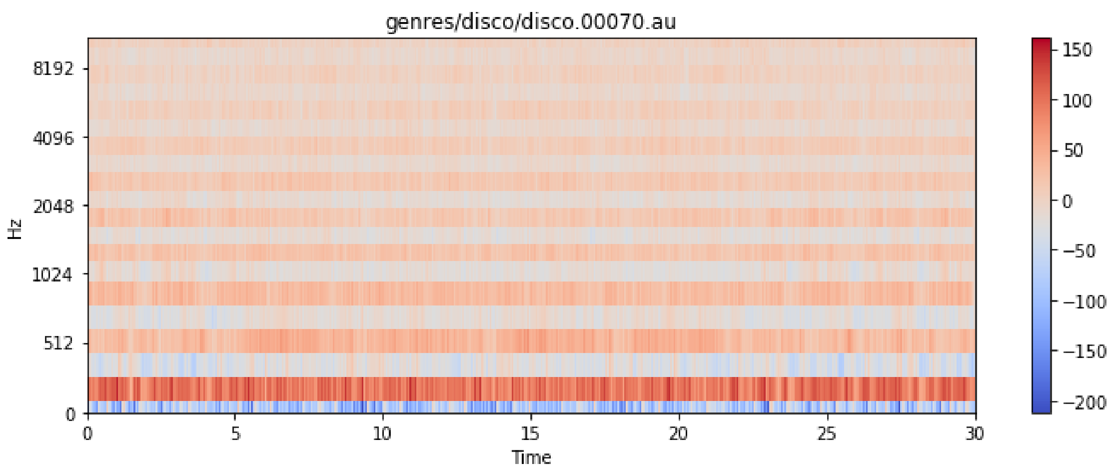

```
In [12]: display_mfcc('whistling.wav')
```



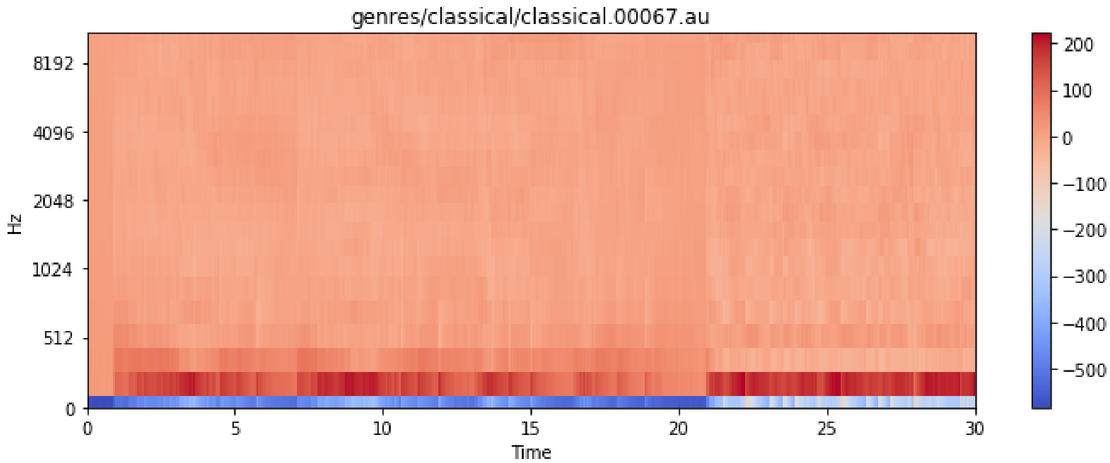

```
In [13]: display_mfcc('genres/disco/disco.00035.au')
```



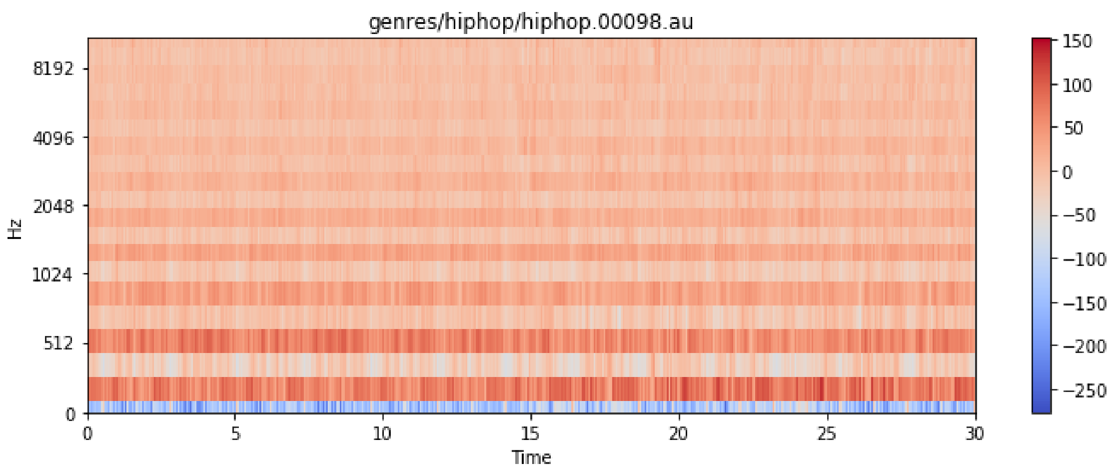
In [14]: display_mfcc('genres/disco/disco.00070.au')



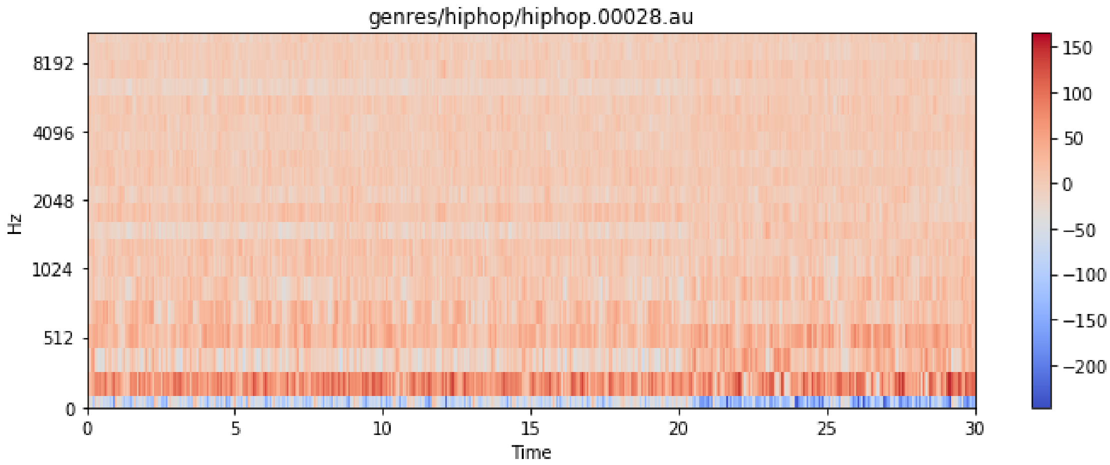
In [15]: display_mfcc('genres/classical/classical.00067.au')




```
In [16]: display_mfcc('genres/hiphop/hiphop.00098.au')
```




```
In [17]: display_mfcc('genres/hiphop/hiphop.00028.au')
```




```
In [2]: def extract_features_song(f):
            y, _=librosa.load(f)
            # get Mel-frequency cepstral  coefficients
            mfcc = librosa.feature.mfcc(y)
            # normalize values between -1, 1 (divide by max)
            mfcc /= np.amax(np.absolute(mfcc))

            return np.ndarray.flatten(mfcc)[:25000]
```

```
In [3]: def generate_features_and_labels():
            all_features = []
            all_labels = []

            genres = ['blues', 'classical', 'country', 'hiphop', 'jazz', 'metal', 'pop', 'regga
            for genre in genres:
                sound_files = glob.glob('genres/'+genre+'/*.au')
                print('Processing %d songs in %s genre...' % (len(sound_files), genre))
                for f in sound_files:
                    features = extract_features_song(f)
                    all_features.append(features)
                    all_labels.append(genre)

            # convert labels to one-hot encoding
            label_uniq_ids, label_row_ids = np.unique(all_labels, return_inverse=True)
            label_row_ids = label_row_ids.astype(np.int32, copy=False)
            onehot_labels = to_categorical(label_row_ids, len(label_uniq_ids))
            return np.stack(all_features), onehot_labels
```

```
In [4]: features, labels = generate_features_and_labels()
```

```
Processing 100 songs in blues genre...
Processing 100 songs in classical genre...
```

```
Processing 100 songs in country genre...
Processing 100 songs in hiphop genre...
Processing 100 songs in jazz genre...
Processing 100 songs in metal genre...
Processing 100 songs in pop genre...
Processing 100 songs in reggae genre...
Processing 100 songs in rock genre...
```

```
In [5]: print(np.shape(features))
        print(np.shape(labels))

        training_split = 0.8

        # last column has genres, turn it in to unique ids
        alldata = np.column_stack((features, labels))

        np.random.shuffle(alldata)
        splitidx = int(len(alldata) * training_split)
        train, test = alldata[:splitidx,:], alldata[splitidx:, :]

        print(np.shape(train))
        print(np.shape(test))

        train_input = train[:, :-10]
        train_labels = train[:,-10:]

        test_input = test[:, :-10]
        test_labels = test[:, -10:]

        print(np.shape(train_input))
        print(np.shape(train_labels))
```

```
(900, 25000)
(900, 9)
(720, 25009)
(180, 25009)
(720, 24999)
(720, 10)
```

```
In [6]: model = Sequential([
            Dense(100, input_dim=np.shape(train_input)[1]),
            Activation('relu'),
            Dense(10),
            Activation('softmax'),
        ])
```

```python
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

print(model.summary())

model.fit(train_input, train_labels, validation_split=0.2, epochs=10, batch_size=32)

loss, acc = model.evaluate(test_input, test_labels, batch_size=32)

print("Done!")
print("Loss: %.4f, accuracy: %.4f" % (loss, acc))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 100)               2500000
_________________________________________________________________
activation_1 (Activation)    (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 2,501,010
Trainable params: 2,501,010
Non-trainable params: 0
_________________________________________________________________
None
Train on 576 samples, validate on 144 samples
Epoch 1/10
576/576 [==============================] - 7s 12ms/step - loss: 1.8817 - acc: 0.3281 - val_loss
Epoch 2/10
576/576 [==============================] - 0s 413us/step - loss: 1.1498 - acc: 0.5990 - val_los
Epoch 3/10
576/576 [==============================] - 0s 385us/step - loss: 0.8766 - acc: 0.6823 - val_los
Epoch 4/10
576/576 [==============================] - 0s 399us/step - loss: 0.6533 - acc: 0.7899 - val_los
Epoch 5/10
576/576 [==============================] - 0s 415us/step - loss: 0.4674 - acc: 0.8819 - val_los
Epoch 6/10
576/576 [==============================] - 0s 420us/step - loss: 0.3602 - acc: 0.9097 - val_los
Epoch 7/10
576/576 [==============================] - 0s 422us/step - loss: 0.2337 - acc: 0.9601 - val_los
Epoch 8/10
576/576 [==============================] - 0s 418us/step - loss: 0.1664 - acc: 0.9878 - val_los
Epoch 9/10
576/576 [==============================] - 0s 411us/step - loss: 0.1013 - acc: 0.9948 - val_los
```

```
Epoch 10/10
576/576 [==============================] - 0s 434us/step - loss: 0.0652 - acc: 1.0000 - val_los
180/180 [==============================] - 0s 189us/step
Done!
Loss: 1.2868, accuracy: 0.6389
```

```
In [7]: import pandas as pd
        from keras.preprocessing.text import Tokenizer
        import numpy as np
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Activation
        from keras.utils import np_utils
        from sklearn.model_selection import StratifiedKFold
```

```
In [8]: d=pd.concat([pd.read_csv("youtube-spam/Youtube01-Psy.csv"),
                     pd.read_csv("youtube-spam/Youtube02-KatyPerry.csv"),
                     pd.read_csv("youtube-spam/Youtube03-LMFAO.csv"),
                     pd.read_csv("youtube-spam/Youtube04-Eminem.csv"),
                     pd.read_csv("youtube-spam/Youtube05-Shakira.csv")])

        d = d.sample(frac=1)
```

```
In [9]: kfold = StratifiedKFold(n_splits=5)
        splits = kfold.split(d, d['CLASS'])
```

```
In [10]: for train, test in splits:
             print("Split")
             print(test)
```

```
Split
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
```

```
 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
 378 379 380 381 382 383 384 385 386 387 388 392 393 394]
Split
[389 390 391 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409
 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427
 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445
 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463
 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481
 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499
 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517
 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535
 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553
 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571
 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589
 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607
 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625
 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643
 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661
 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679
 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697
 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715
 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733
 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751
 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769
 770 771 774 775 776 778 780 785 786 789 793 795 797]
Split
[ 772  773  777  779  781  782  783  784  787  788  790  791  792  794
  796  798  799  800  801  802  803  804  805  806  807  808  809  810
  811  812  813  814  815  816  817  818  819  820  821  822  823  824
  825  826  827  828  829  830  831  832  833  834  835  836  837  838
  839  840  841  842  843  844  845  846  847  848  849  850  851  852
  853  854  855  856  857  858  859  860  861  862  863  864  865  866
  867  868  869  870  871  872  873  874  875  876  877  878  879  880
  881  882  883  884  885  886  887  888  889  890  891  892  893  894
  895  896  897  898  899  900  901  902  903  904  905  906  907  908
  909  910  911  912  913  914  915  916  917  918  919  920  921  922
  923  924  925  926  927  928  929  930  931  932  933  934  935  936
  937  938  939  940  941  942  943  944  945  946  947  948  949  950
  951  952  953  954  955  956  957  958  959  960  961  962  963  964
  965  966  967  968  969  970  971  972  973  974  975  976  977  978
  979  980  981  982  983  984  985  986  987  988  989  990  991  992
  993  994  995  996  997  998  999 1000 1001 1002 1003 1004 1005 1006
 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020
 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034
 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048
 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062
```

```
 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076
 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090
 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104
 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118
 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132
 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146
 1147 1149 1150 1152 1153 1154 1156 1157 1158 1159 1160 1161 1162 1164
 1166 1168 1169 1171 1172 1173 1174 1175 1178 1180 1183 1185 1188]
Split
[1148 1151 1155 1163 1165 1167 1170 1176 1177 1179 1181 1182 1184 1186
 1187 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201
 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215
 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229
 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243
 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257
 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271
 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285
 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299
 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313
 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327
 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338 1339 1340 1341
 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352 1353 1354 1355
 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369
 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383
 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397
 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411
 1412 1413 1414 1415 1416 1417 1418 1419 1420 1421 1422 1423 1424 1425
 1426 1427 1428 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439
 1440 1441 1442 1443 1444 1445 1446 1447 1448 1449 1450 1451 1452 1453
 1454 1455 1456 1457 1458 1459 1460 1461 1462 1463 1464 1465 1466 1467
 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481
 1482 1483 1484 1485 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495
 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509
 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519 1520 1521 1522 1523
 1524 1525 1526 1527 1528 1529 1530 1531 1532 1533 1534 1535 1536 1537
 1538 1539 1540 1541 1542 1543 1544 1545 1546 1547 1548 1549 1550 1555
 1557 1559 1560 1561 1563 1565 1566 1567 1570 1571 1572 1573 1575]
Split
[1551 1552 1553 1554 1556 1558 1562 1564 1568 1569 1574 1576 1577 1578
 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591 1592
 1593 1594 1595 1596 1597 1598 1599 1600 1601 1602 1603 1604 1605 1606
 1607 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620
 1621 1622 1623 1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634
 1635 1636 1637 1638 1639 1640 1641 1642 1643 1644 1645 1646 1647 1648
 1649 1650 1651 1652 1653 1654 1655 1656 1657 1658 1659 1660 1661 1662
 1663 1664 1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676
 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690
 1691 1692 1693 1694 1695 1696 1697 1698 1699 1700 1701 1702 1703 1704
```

```
1705 1706 1707 1708 1709 1710 1711 1712 1713 1714 1715 1716 1717 1718
1719 1720 1721 1722 1723 1724 1725 1726 1727 1728 1729 1730 1731 1732
1733 1734 1735 1736 1737 1738 1739 1740 1741 1742 1743 1744 1745 1746
1747 1748 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760
1761 1762 1763 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788
1789 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802
1803 1804 1805 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816
1817 1818 1819 1820 1821 1822 1823 1824 1825 1826 1827 1828 1829 1830
1831 1832 1833 1834 1835 1836 1837 1838 1839 1840 1841 1842 1843 1844
1845 1846 1847 1848 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858
1859 1860 1861 1862 1863 1864 1865 1866 1867 1868 1869 1870 1871 1872
1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886
1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900
1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914
1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928
1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942
1943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955]
```

In [23]:
```python
def train_and_test(train_idx, test_idx):
    train_content = d['CONTENT'].iloc[train_idx]
    test_content = d['CONTENT'].iloc[test_idx]

    tokenizer = Tokenizer(num_words=2000)

    # Learn the training words (not the testing words!)
    tokenizer.fit_on_texts(train_content)

    # options for made : binary, freq, tfidf
    d_train_inputs = tokenizer.texts_to_matrix(train_content, mode='tfidf')
    d_test_inputs = tokenizer.texts_to_matrix(test_content, mode='tfidf')

    # divide tfidf by max
    d_train_inputs = d_train_inputs/np.amax(np.absolute(d_train_inputs))
    d_test_inputs = d_test_inputs/np.amax(np.absolute(d_test_inputs))

    # substract mean, to get values between -1 and 1
    d_train_inputs = d_train_inputs - np.mean(d_train_inputs)
    d_test_inputs = d_test_inputs - np.mean(d_test_inputs)

    # one-hot encoding of outputs
    d_train_outputs = np_utils.to_categorical(d['CLASS'].iloc[train_idx])
    d_test_outputs = np_utils.to_categorical(d['CLASS'].iloc[test_idx])

    model = Sequential()
    model.add(Dense(512, input_shape=(2000,)))
    model.add(Activation('relu'))
```

```
    model.add(Dropout(0.5))
    model.add(Dense(2))
    model.add(Activation('softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adamax',
                  metrics=['accuracy'])

    model.fit(d_train_inputs, d_train_outputs, epochs=10, batch_size=16)
    scores = model.evaluate(d_test_inputs, d_test_outputs)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    return scores
```

In [24]:
```
kfold = StratifiedKFold(n_splits=5)
splits = kfold.split(d, d['CLASS'])
cvscores = []
for train_idx, test_idx in splits:
    scores = train_and_test(train_idx, test_idx)
    cvscores.append(scores[1]*100)
```

```
Epoch 1/10
1564/1564 [==============================] - 1s 857us/step - loss: 0.5809 - acc: 0.8350
Epoch 2/10
1564/1564 [==============================] - 1s 350us/step - loss: 0.3361 - acc: 0.9137
Epoch 3/10
1564/1564 [==============================] - 1s 322us/step - loss: 0.2136 - acc: 0.9425
Epoch 4/10
1564/1564 [==============================] - 1s 327us/step - loss: 0.1611 - acc: 0.9540
Epoch 5/10
1564/1564 [==============================] - 1s 333us/step - loss: 0.1317 - acc: 0.9616
Epoch 6/10
1564/1564 [==============================] - 1s 332us/step - loss: 0.1125 - acc: 0.9668
Epoch 7/10
1564/1564 [==============================] - 1s 326us/step - loss: 0.0984 - acc: 0.9725
Epoch 8/10
1564/1564 [==============================] - 1s 325us/step - loss: 0.0844 - acc: 0.9757
Epoch 9/10
1564/1564 [==============================] - 0s 311us/step - loss: 0.0785 - acc: 0.9751
Epoch 10/10
1564/1564 [==============================] - 1s 331us/step - loss: 0.0699 - acc: 0.9802
392/392 [==============================] - 0s 270us/step
acc: 96.17%
Epoch 1/10
1565/1565 [==============================] - 1s 649us/step - loss: 0.5975 - acc: 0.7942
Epoch 2/10
1565/1565 [==============================] - 1s 347us/step - loss: 0.3677 - acc: 0.9137
Epoch 3/10
1565/1565 [==============================] - 1s 360us/step - loss: 0.2272 - acc: 0.9412
Epoch 4/10
```

```
1565/1565 [==============================] - 1s 361us/step - loss: 0.1651 - acc: 0.9540
Epoch 5/10
1565/1565 [==============================] - 1s 357us/step - loss: 0.1398 - acc: 0.9610
Epoch 6/10
1565/1565 [==============================] - 1s 332us/step - loss: 0.1168 - acc: 0.9661
Epoch 7/10
1565/1565 [==============================] - 0s 315us/step - loss: 0.1007 - acc: 0.9725
Epoch 8/10
1565/1565 [==============================] - 1s 337us/step - loss: 0.0893 - acc: 0.9725
Epoch 9/10
1565/1565 [==============================] - 1s 367us/step - loss: 0.0818 - acc: 0.9776
Epoch 10/10
1565/1565 [==============================] - 1s 373us/step - loss: 0.0761 - acc: 0.9744
391/391 [==============================] - 0s 314us/step
acc: 95.40%
Epoch 1/10
1565/1565 [==============================] - 1s 688us/step - loss: 0.5896 - acc: 0.7936
Epoch 2/10
1565/1565 [==============================] - 1s 402us/step - loss: 0.3609 - acc: 0.9093
Epoch 3/10
1565/1565 [==============================] - 1s 404us/step - loss: 0.2288 - acc: 0.9438
Epoch 4/10
1565/1565 [==============================] - 1s 396us/step - loss: 0.1692 - acc: 0.9514
Epoch 5/10
1565/1565 [==============================] - 1s 338us/step - loss: 0.1383 - acc: 0.9604
Epoch 6/10
1565/1565 [==============================] - 1s 351us/step - loss: 0.1158 - acc: 0.9693
Epoch 7/10
1565/1565 [==============================] - 1s 374us/step - loss: 0.1030 - acc: 0.9712
Epoch 8/10
1565/1565 [==============================] - 1s 366us/step - loss: 0.0877 - acc: 0.9744
Epoch 9/10
1565/1565 [==============================] - 1s 372us/step - loss: 0.0838 - acc: 0.9757
Epoch 10/10
1565/1565 [==============================] - 1s 337us/step - loss: 0.0724 - acc: 0.9783
391/391 [==============================] - 0s 338us/step
acc: 94.37%
Epoch 1/10
1565/1565 [==============================] - 1s 705us/step - loss: 0.5980 - acc: 0.7853
Epoch 2/10
1565/1565 [==============================] - 1s 348us/step - loss: 0.3655 - acc: 0.9208
Epoch 3/10
1565/1565 [==============================] - 1s 351us/step - loss: 0.2304 - acc: 0.9508
Epoch 4/10
1565/1565 [==============================] - 1s 347us/step - loss: 0.1695 - acc: 0.9604
Epoch 5/10
1565/1565 [==============================] - 1s 373us/step - loss: 0.1398 - acc: 0.9636
Epoch 6/10
```

```
1565/1565 [==============================] - 1s 365us/step - loss: 0.1183 - acc: 0.9668
Epoch 7/10
1565/1565 [==============================] - 1s 365us/step - loss: 0.1014 - acc: 0.9719
Epoch 8/10
1565/1565 [==============================] - 1s 376us/step - loss: 0.0941 - acc: 0.9738
Epoch 9/10
1565/1565 [==============================] - 1s 360us/step - loss: 0.0822 - acc: 0.9764
Epoch 10/10
1565/1565 [==============================] - 1s 390us/step - loss: 0.0734 - acc: 0.9815
391/391 [==============================] - 0s 371us/step
acc: 94.63%
Epoch 1/10
1565/1565 [==============================] - 1s 665us/step - loss: 0.6116 - acc: 0.7278
Epoch 2/10
1565/1565 [==============================] - 1s 355us/step - loss: 0.3895 - acc: 0.9220
Epoch 3/10
1565/1565 [==============================] - 1s 333us/step - loss: 0.2435 - acc: 0.9412
Epoch 4/10
1565/1565 [==============================] - 1s 338us/step - loss: 0.1797 - acc: 0.9534
Epoch 5/10
1565/1565 [==============================] - 1s 346us/step - loss: 0.1423 - acc: 0.9585
Epoch 6/10
1565/1565 [==============================] - 1s 335us/step - loss: 0.1242 - acc: 0.9642
Epoch 7/10
1565/1565 [==============================] - 1s 339us/step - loss: 0.1042 - acc: 0.9693
Epoch 8/10
1565/1565 [==============================] - 1s 337us/step - loss: 0.0925 - acc: 0.9764
Epoch 9/10
1565/1565 [==============================] - 1s 337us/step - loss: 0.0874 - acc: 0.9751
Epoch 10/10
1565/1565 [==============================] - 1s 339us/step - loss: 0.0749 - acc: 0.9815
391/391 [==============================] - 0s 384us/step
acc: 95.40%
```

In [ ]: