```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Import common libraries"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {},
   "outputs": [
    {
     "name": "stderr",
     "output_type": "stream",
     "text": [
      "C:\\Users\\User\\anaconda3\\lib\\site-packages\\numpy\\
_distributor_init.py:32: UserWarning: loaded more than 1 DLL from .libs:\n",
      "C:\\Users\\User\\anaconda3\\lib\\site-packages\\numpy\\.libs\\
libopenblas.NOIJJG62EMASZI6NYURL6JBKM4EVBGM7.gfortran-win_amd64.dll\n",
      "C:\\Users\\User\\anaconda3\\lib\\site-packages\\numpy\\.libs\\
libopenblas.PYQHXLVVQ7VESDPUVUADXEVJOBGHJPAY.gfortran-win_amd64.dll\n",
      "  stacklevel=1)\n"
     ]
    }
   ],
   "source": [
    "import tensorflow as tf\n",
    "import numpy as np\n",
    "import pandas as pd\n",
    "import random as rn\n",
    "from model_persistance import ModelPersistance\n",
    "from evaluate_classification import EvaluateBinaryClassification"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Initialise Random variables"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {
    "slideshow": {
     "slide_type": "slide"
    }
   },
   "outputs": [],
   "source": [
    "SEED = 123\n",
    "np.random.seed(SEED)\n",
    "tf.random.set_seed(SEED)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Loading Data"
```

```
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "metadata": {},
   "outputs": [],
   "source": [
    "BASE = 'D:\\\\ResearchDataGtx1060\\\\SentimentData\\\\Hate\\\\'\n",
    "fins_train = ['random_hate_train.csv']\n",
    "fins_test = ['eastasian_hate_test.csv']\n",
    "track = 0"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "metadata": {},
   "outputs": [],
   "source": [
    "# We apply only this preprocessing because our data is already
preprocessed\n",
    "def cleanNonAscii(text):\n",
    "    '''\n",
    "    Remove Non ASCII characters from the dataset.\n",
    "    Arguments:\n",
    "        text: str\n",
    "    returns: \n",
    "        text: str\n",
    "    '''\n",
    "    return ''.join(i for i in text if ord(i) < 128)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/html": [
       "<div>\n",
       "<style scoped>\n",
       "    .dataframe tbody tr th:only-of-type {\n",
       "        vertical-align: middle;\n",
       "    }\n",
       "\n",
       "    .dataframe tbody tr th {\n",
       "        vertical-align: top;\n",
       "    }\n",
       "\n",
       "    .dataframe thead th {\n",
       "        text-align: right;\n",
       "    }\n",
       "</style>\n",
       "<table border=\"1\" class=\"dataframe\">\n",
       "  <thead>\n",
       "    <tr style=\"text-align: right;\">\n",
       "      <th></th>\n",
       "      <th>label</th>\n",
       "      <th>text</th>\n",
       "    </tr>\n",
       "  </thead>\n",
       "  <tbody>\n",
```

```
       "    <tr>\n",
       "      <th>0</th>\n",
       "      <td>1</td>\n",
       "      <td>&lt;user&gt; if you are one of the &lt;number&gt; mil
&lt;has...</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>1</th>\n",
       "      <td>0</td>\n",
       "      <td>best &lt;hashtag&gt; law of attraction &lt;/hashtag&gt;
&lt;h...</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>2</th>\n",
       "      <td>1</td>\n",
       "      <td>&lt;hashtag&gt; michelle obama &lt;/hashtag&gt; is the
mos...</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>3</th>\n",
       "      <td>0</td>\n",
       "      <td>smiling because life is good rite now ! &lt;repea...</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>4</th>\n",
       "      <td>0</td>\n",
       "      <td>Ã£ Â¢ Ã¢ Â Ã¢ Â¤ Ã£ Â¯ Ã¢ Â¸ Ã¢ Â Ã£ Â¢ Ã¢ Â Ã¢ Â¤ Ã£ Â¯ Ã¢ Â¸
Ã¢ ...</td>\n",
       "    </tr>\n",
       "  </tbody>\n",
       "</table>\n",
       "</div>"
      ],
      "text/plain": [
       "   label                                                text\n",
       "0      1  <user> if you are one of the <number> mil <has...\n",
       "1      0  best <hashtag> law of attraction </hashtag> <h...\n",
       "2      1  <hashtag> michelle obama </hashtag> is the mos...\n",
       "3      0  smiling because life is good rite now ! <repea...\n",
       "4      0  Ã£ Â¢ Ã¢ Â Ã¢ Â¤ Ã£ Â¯ Ã¢ Â¸ Ã¢ Â Ã£ Â¢ Ã¢ Â Ã¢ Â¤ Ã£ Â¯ Ã¢ Â¸
Ã¢ ..."
      ]
     },
     "execution_count": 5,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "df_train = pd.read_csv(BASE+fins_train[track])\n",
    "#df.columns=['label', 'text']\n",
    "df_train.head()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/html": [
       "<div>\n",
       "<style scoped>\n",
```

```
      "    .dataframe tbody tr th:only-of-type {\n",
      "        vertical-align: middle;\n",
      "    }\n",
      "\n",
      "    .dataframe tbody tr th {\n",
      "        vertical-align: top;\n",
      "    }\n",
      "\n",
      "    .dataframe thead th {\n",
      "        text-align: right;\n",
      "    }\n",
      "</style>\n",
      "<table border=\"1\" class=\"dataframe\">\n",
      "  <thead>\n",
      "    <tr style=\"text-align: right;\">\n",
      "      <th></th>\n",
      "      <th>text</th>\n",
      "    </tr>\n",
      "    <tr>\n",
      "      <th>label</th>\n",
      "      <th></th>\n",
      "    </tr>\n",
      "  </thead>\n",
      "  <tbody>\n",
      "    <tr>\n",
      "      <th>0</th>\n",
      "      <td>2242</td>\n",
      "    </tr>\n",
      "    <tr>\n",
      "      <th>1</th>\n",
      "      <td>2242</td>\n",
      "    </tr>\n",
      "  </tbody>\n",
      "</table>\n",
      "</div>"
     ],
     "text/plain": [
      "       text\n",
      "label      \n",
      "0      2242\n",
      "1      2242"
     ]
    },
    "execution_count": 6,
    "metadata": {},
    "output_type": "execute_result"
   }
  ],
  "source": [
   "df_train.groupby('label').count()"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 7,
  "metadata": {},
  "outputs": [
   {
    "data": {
     "text/plain": [
      "4484"
     ]
    },
    "execution_count": 7,
```

```
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "len(df_train)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 8,
   "metadata": {},
   "outputs": [],
   "source": [
    "df_train['text'] = df_train['text'].apply(cleanNonAscii)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 9,
   "metadata": {},
   "outputs": [],
   "source": [
    "X_train, y_train = df_train['text'].values, df_train['label'].values"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 10,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/html": [
       "<div>\n",
       "<style scoped>\n",
       "    .dataframe tbody tr th:only-of-type {\n",
       "        vertical-align: middle;\n",
       "    }\n",
       "\n",
       "    .dataframe tbody tr th {\n",
       "        vertical-align: top;\n",
       "    }\n",
       "\n",
       "    .dataframe thead th {\n",
       "        text-align: right;\n",
       "    }\n",
       "</style>\n",
       "<table border=\"1\" class=\"dataframe\">\n",
       "  <thead>\n",
       "    <tr style=\"text-align: right;\">\n",
       "      <th></th>\n",
       "      <th>label</th>\n",
       "      <th>text</th>\n",
       "    </tr>\n",
       "  </thead>\n",
       "  <tbody>\n",
       "    <tr>\n",
       "      <th>0</th>\n",
       "      <td>1</td>\n",
       "      <td>&amp;lt;user&amp;gt; &amp;lt;user&amp;gt; the chinese are probably sprayin...</td>\n",
       "    </tr>\n",
       "    <tr>\n",
```

```
       "      <th>1</th>\n",
       "      <td>0</td>\n",
       "      <td>rt &lt;user&gt; : unpatriotic losers are tweeting ou...</td>\
n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>2</th>\n",
       "      <td>1</td>\n",
       "      <td>&lt;user&gt; thus &lt;hashtag&gt; 2019 n co v &lt;/hashtag&gt;
i...</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>3</th>\n",
       "      <td>0</td>\n",
       "      <td>north korea closes borders to avoid coronaviru...</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>4</th>\n",
       "      <td>1</td>\n",
       "      <td>&lt;user&gt; this is a declaration of war . it prove...</td>\
n",
       "    </tr>\n",
       "  </tbody>\n",
       "</table>\n",
       "</div>"
      ],
      "text/plain": [
       "   label                                               text\n",
       "0      1  <user> <user> the chinese are probably sprayin...\n",
       "1      0  rt <user> : unpatriotic losers are tweeting ou...\n",
       "2      1  <user> thus <hashtag> 2019 n co v </hashtag> i...\n",
       "3      0  north korea closes borders to avoid coronaviru...\n",
       "4      1  <user> this is a declaration of war . it prove..."
      ]
     },
     "execution_count": 10,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "df_test = pd.read_csv(BASE+fins_test[track])\n",
    "#df_test.columns = ['label', 'text']\n",
    "df_test.head()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 11,
   "metadata": {},
   "outputs": [],
   "source": [
    "df_test['text'] = df_test['text'].apply(cleanNonAscii)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 12,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/html": [
       "<div>\n",
```

```
       "<style scoped>\n",
       "    .dataframe tbody tr th:only-of-type {\n",
       "        vertical-align: middle;\n",
       "    }\n",
       "\n",
       "    .dataframe tbody tr th {\n",
       "        vertical-align: top;\n",
       "    }\n",
       "\n",
       "    .dataframe thead th {\n",
       "        text-align: right;\n",
       "    }\n",
       "</style>\n",
       "<table border=\"1\" class=\"dataframe\">\n",
       "  <thead>\n",
       "    <tr style=\"text-align: right;\">\n",
       "      <th></th>\n",
       "      <th>label</th>\n",
       "      <th>text</th>\n",
       "    </tr>\n",
       "  </thead>\n",
       "  <tbody>\n",
       "    <tr>\n",
       "      <th>0</th>\n",
       "      <td>1</td>\n",
       "      <td>&lt;user&gt; &lt;user&gt; the chinese are probably
sprayin...</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>1</th>\n",
       "      <td>0</td>\n",
       "      <td>rt &lt;user&gt; : unpatriotic losers are tweeting ou...</td>\
n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>2</th>\n",
       "      <td>1</td>\n",
       "      <td>&lt;user&gt; thus &lt;hashtag&gt; 2019 n co v &lt;/hashtag&gt;
i...</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>3</th>\n",
       "      <td>0</td>\n",
       "      <td>north korea closes borders to avoid coronaviru...</td>\n",
       "    </tr>\n",
       "    <tr>\n",
       "      <th>4</th>\n",
       "      <td>1</td>\n",
       "      <td>&lt;user&gt; this is a declaration of war . it prove...</td>\
n",
       "    </tr>\n",
       "  </tbody>\n",
       "</table>\n",
       "</div>"
      ],
      "text/plain": [
       "   label                                               text\n",
       "0      1  <user> <user> the chinese are probably sprayin...\n",
       "1      0  rt <user> : unpatriotic losers are tweeting ou...\n",
       "2      1  <user> thus <hashtag> 2019 n co v </hashtag> i...\n",
       "3      0  north korea closes borders to avoid coronaviru...\n",
       "4      1  <user> this is a declaration of war . it prove..."
      ]
     },
```

```
   "execution_count": 12,
   "metadata": {},
   "output_type": "execute_result"
  }
 ],
 "source": [
  "df_test.head()"
 ]
},
{
 "cell_type": "code",
 "execution_count": 13,
 "metadata": {},
 "outputs": [],
 "source": [
  "X_test, y_test = df_test['text'].values, df_test['label'].values"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "# Transforming data suitable for model format"
 ]
},
{
 "cell_type": "code",
 "execution_count": 14,
 "metadata": {},
 "outputs": [],
 "source": [
  "# X_new = []\n",
  "# X_new.extend(X_train)\n",
  "# X_new.extend(X_test)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 15,
 "metadata": {},
 "outputs": [],
 "source": [
  "from keras.preprocessing.text import Tokenizer\n",
  "from keras.preprocessing.sequence import pad_sequences\n",
  "num_words = 100000\n",
  "tokenizer = Tokenizer(num_words=num_words)\n",
  "tokenizer.fit_on_texts(X_train)\n",
  "xtrain = tokenizer.texts_to_sequences(X_train)\n",
  "maxlen = max(map(lambda x: len(x),xtrain))\n",
  "xtrain = pad_sequences(xtrain, maxlen=maxlen)\n",
  "\n",
  "xtest = tokenizer.texts_to_sequences(X_test)\n",
  "xtest = pad_sequences(xtest, maxlen=maxlen)"
 ]
},
{
 "cell_type": "markdown",
 "metadata": {},
 "source": [
  "### Loading word embedding and mapping data to that word embedding"
 ]
},
{
 "cell_type": "code",
```

```
   "execution_count": 16,
   "metadata": {},
   "outputs": [],
   "source": [
    "from gensim.models import KeyedVectors\n",
    "W2V_BASE = 'D:\\\\\\ResearchDataGtx1060\\\\\TwitterDataAustralia\\\\\\\
W2V_AusTweets_200d_MinCount100\\\\\'\n",
    "model_ug_cbow = KeyedVectors.load(W2V_BASE+'vectors.txt')\n",
    "\n",
    "# W2V_BASE = 'D:\\\\\ResearchDataGtx1060\\\\\HASOC2020Datasets\\\\\eng\\\\\
w2v_sentiTweets_200d_minCount10\\\\\'\n",
    "# model_ug_cbow = KeyedVectors.load(W2V_BASE+'vectors.txt')\n",
    "\n",
    "embeddings_index = {}\n",
    "for w in model_ug_cbow.wv.vocab.keys():\n",
    "    embeddings_index[w] = model_ug_cbow.wv[w]\n",
    "\n",
    "embedding_matrix = np.zeros((num_words, 200))\n",
    "for word, i in tokenizer.word_index.items():\n",
    "    if i >= num_words:\n",
    "        continue\n",
    "    embedding_vector = embeddings_index.get(word)\n",
    "    if embedding_vector is not None:\n",
    "        embedding_matrix[i] = embedding_vector"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 30,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "array([[   0,    0,    0, ..., 1254,   21, 1255],\n",
       "       [   0,    0,    0, ...,    7, 2860,    1],\n",
       "       [   0,    0,    0, ...,    1, 4420,    1],\n",
       "       ...,\n",
       "       [   0,    0,    0, ...,  214,  207,    1],\n",
       "       [   0,    0,    0, ..., 3822,  259, 2035],\n",
       "       [   0,    0,    0, ...,   42,  465,    1]])"
      ]
     },
     "execution_count": 30,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "xtrain"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Creating CNN model and training it for n epocs"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 17,
   "metadata": {},
   "outputs": [
```

```
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "loading word vectors\n",
      "Epoch 1/3\n",
      "WARNING:tensorflow:AutoGraph could not transform <function
Model.make_train_function.<locals>.train_function at 0x0000024ED0261798> and
will run it as-is.\n",
      "Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.\n",
      "Cause: 'arguments' object has no attribute 'posonlyargs'\n",
      "To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert\n",
      "WARNING: AutoGraph could not transform <function
Model.make_train_function.<locals>.train_function at 0x0000024ED0261798> and
will run it as-is.\n",
      "Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.\n",
      "Cause: 'arguments' object has no attribute 'posonlyargs'\n",
      "To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert\n",
      "141/141 [==============================] - 49s 345ms/step - loss: 0.5707
- accuracy: 0.6624\n",
      "Epoch 2/3\n",
      "141/141 [==============================] - 51s 360ms/step - loss: 0.2194
- accuracy: 0.9168\n",
      "Epoch 3/3\n",
      "141/141 [==============================] - 54s 383ms/step - loss: 0.0913
- accuracy: 0.9686\n"
     ]
    },
    {
     "data": {
      "text/plain": [
       "<tensorflow.python.keras.callbacks.History at 0x24ecfed5388>"
      ]
     },
     "execution_count": 17,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "from keras.layers import Dense, Dropout\n",
    "from keras.layers.embeddings import Embedding\n",
    "from keras.layers import Conv1D, GlobalMaxPooling1D\n",
    "from keras.layers import Input, concatenate, Activation\n",
    "from keras.models import Model\n",
    "\n",
    "def create_cnn_model():\n",
    "    tweet_input = Input(shape=(maxlen,), dtype='int32')\n",
    "    \n",
    "    print('loading word vectors')\n",
    "    #tweet_encoder = Embedding(num_words, 200, weights=[embedding_matrix],
input_length=maxlen, trainable=True)(tweet_input)\n",
    "    tweet_encoder = Embedding(num_words, 200, input_length=maxlen,
trainable=True)(tweet_input)    \n",
    "    tweet_encoder = Dropout(0.5)(tweet_encoder)\n",
    "    \n",
    "    bigram_branch = Conv1D(filters=128, kernel_size=3, padding='valid',
activation='relu', strides=1)(tweet_encoder)\n",
```

```
    "    bigram_branch = GlobalMaxPooling1D()(bigram_branch)\n",
    "    bigram_branch = Dropout(0.5)(bigram_branch)\n",
    "    \n",
    "    trigram_branch = Conv1D(filters=256, kernel_size=4, padding='valid',
activation='relu', strides=1)(tweet_encoder)\n",
    "    trigram_branch = GlobalMaxPooling1D()(trigram_branch)\n",
    "    trigram_branch = Dropout(0.2)(trigram_branch)\n",
    "    \n",
    "    fourgram_branch = Conv1D(filters=512, kernel_size=5, padding='valid',
activation='relu', strides=1)(tweet_encoder)\n",
    "    fourgram_branch = GlobalMaxPooling1D()(fourgram_branch)\n",
    "    fourgram_branch = Dropout(0.2)(fourgram_branch)\n",
    "    \n",
    "    merged = concatenate([bigram_branch, trigram_branch, fourgram_branch],
axis=1)\n",
    "\n",
    "    merged = Dense(256, activation='relu')(merged)\n",
    "    merged = Dropout(0.5)(merged)\n",
    "    \n",
    "    merged = Dense(1)(merged)\n",
    "    output = Activation('sigmoid')(merged)\n",
    "    \n",
    "    model = Model(inputs=[tweet_input], outputs=[output])\n",
    "    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])\n",
    "    #model.summary()\n",
    "    return model\n",
    "\n",
    "cnn_model = create_cnn_model()\n",
    "cnn_model.fit(xtrain, y_train, epochs=3, batch_size=32, verbose=1)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Evaluating the model with test dataset"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 18,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "WARNING:tensorflow:AutoGraph could not transform <function
Model.make_predict_function.<locals>.predict_function at 0x0000024ED0E80288> and
will run it as-is.\n",
      "Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.\n",
      "Cause: 'arguments' object has no attribute 'posonlyargs'\n",
      "To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert\n",
      "WARNING: AutoGraph could not transform <function
Model.make_predict_function.<locals>.predict_function at 0x0000024ED0E80288> and
will run it as-is.\n",
      "Please report this to the TensorFlow team. When filing the bug, set the
verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full
output.\n",
      "Cause: 'arguments' object has no attribute 'posonlyargs'\n",
```

```
      "To silence this warning, decorate the function with
@tf.autograph.experimental.do_not_convert\n",
      "244/244 [==============================] - 3s 14ms/step\n",
      "EvaluateBinaryClassification Object Created\n",
      "\n",
      "Total Samples\t7796\n",
      "Positive Samples\t3898\n",
      "Negative Samples\t3898\n",
      "True Positive\t2612\n",
      "True Negative\t1994\n",
      "False Positive\t1904\n",
      "False Negative\t1286\n",
      "Accuracy\t0.5908158029758851\n",
      "Precision\t0.5783879539415412\n",
      "Recall\t0.6700872242175474\n",
      "F1 Measure\t0.6208699786070834\n",
      "Cohen Kappa Score\t0.1816316059517702\n",
      "Area Under Curve\t0.590815802975885\n",
      "\n",
      "              precision    recall  f1-score   support\n",
      "\n",
      "           0       0.61      0.51      0.56      3898\n",
      "           1       0.58      0.67      0.62      3898\n",
      "\n",
      "    accuracy                           0.59      7796\n",
      "   macro avg       0.59      0.59      0.59      7796\n",
      "weighted avg       0.59      0.59      0.59      7796\n",
      "\n"
     ]
    }
   ],
   "source": [
    "p = cnn_model.predict(xtest,verbose=1)\n",
    "predicted = [int(round(x[0])) for x in p]\n",
    "actual = y_test\n",
    "\n",
    "ebc = EvaluateBinaryClassification(gnd_truths = actual, predictions =
predicted)\n",
    "print(ebc.get_full_report())"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 19,
   "metadata": {},
   "outputs": [],
   "source": [
    "ebc.save_full_report(model_name='CNN_no_w2v',
path='domain_adaptation_rerun_randomhate_eastasianhate_')"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Store the trained model"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 20,
   "metadata": {},
   "outputs": [
    {
```

```
     "data": {
      "text/plain": [
       "\"\\nmp = ModelPersistance(store_path = BASE + 'stored_models\\\\\ncnn_w2v_mincount10')\\nmp.store_model(tokenizer=tokenizer, model=cnn_model,\nmax_len=maxlen)\\n\""
      ]
     },
     "execution_count": 20,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "'''\n",
    "mp = ModelPersistance(store_path = BASE + 'stored_models\\\\\ncnn_w2v_mincount10')\n",
    "mp.store_model(tokenizer=tokenizer, model=cnn_model, max_len=maxlen)\n",
    "'''"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Load Stored Model to Predict on Unknown Data"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 21,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "\"\\nmp = ModelPersistance(store_path = BASE + 'stored_models\\\\\ncnn_w2v_mincount10')\\ntokenizer, cnn_model, maxlen = mp.restore_model()\\n\""
      ]
     },
     "execution_count": 21,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "'''\n",
    "mp = ModelPersistance(store_path = BASE + 'stored_models\\\\\ncnn_w2v_mincount10')\n",
    "tokenizer, cnn_model, maxlen = mp.restore_model()\n",
    "'''"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Load Unknown Data and Predict"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 22,
   "metadata": {},
   "outputs": [
```

```
    {
     "data": {
      "text/plain": [
       "\"\\nUNKNOWN_CSV = BASE+'prepro_hasoc_2020_en_test.csv'\\ndf_unk =
pd.read_csv(UNKNOWN_CSV, encoding='utf8')\\ndf_unk.head(5)\\n\""
      ]
     },
     "execution_count": 22,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "'''\n",
    "UNKNOWN_CSV = BASE+'prepro_hasoc_2020_en_test.csv'\n",
    "df_unk = pd.read_csv(UNKNOWN_CSV, encoding='utf8')\n",
    "df_unk.head(5)\n",
    "'''"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 23,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "\"\\nX_unk = list(df_unk['text'].astype(str))\\nxunk =
tokenizer.texts_to_sequences(X_unk)\\nxunk = pad_sequences(xunk,
maxlen=maxlen)\\n#loaded_model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])\\np_unk =
cnn_model.predict(xunk,verbose=0)\\np_unk[:10]\\n\""
      ]
     },
     "execution_count": 23,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "'''\n",
    "X_unk = list(df_unk['text'].astype(str))\n",
    "xunk = tokenizer.texts_to_sequences(X_unk)\n",
    "xunk = pad_sequences(xunk, maxlen=maxlen)\n",
    "#loaded_model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])\n",
    "p_unk = cnn_model.predict(xunk,verbose=0)\n",
    "p_unk[:10]\n",
    "'''"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 24,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "'\\npred_unk = [int(round(x[0])) for x in p_unk]\\npred_unk =
np.array(pred_unk)\\nsum(pred_unk)\\n'"
      ]
     },
```

```
     "execution_count": 24,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "'''\n",
    "pred_unk = [int(round(x[0])) for x in p_unk]\n",
    "pred_unk = np.array(pred_unk)\n",
    "sum(pred_unk)\n",
    "'''"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Store the prediction"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 25,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "\"\\nLANGUAGE = 'EN'\\nSUBTASK_NAME = 'A'\\npred_fname =
'submission_{}_{}.csv'.format(LANGUAGE, SUBTASK_NAME)\\
nBASE+'Predictions\\\\'+pred_fname\\n\""
      ]
     },
     "execution_count": 25,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "'''\n",
    "LANGUAGE = 'EN'\n",
    "SUBTASK_NAME = 'A'\n",
    "pred_fname = 'submission_{}_{}.csv'.format(LANGUAGE, SUBTASK_NAME)\n",
    "BASE+'Predictions\\\\'+pred_fname\n",
    "'''"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 26,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "\"\\ndf_unk[['tweet_id', 'task1', 'ID']]\\n\""
      ]
     },
     "execution_count": 26,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
```

```
    "'''\n",
    "df_unk[['tweet_id', 'task1', 'ID']]\n",
    "'''"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 27,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "\"\\ni2t = ['NOT', 'HOF']\\ndf_unk['task1'] = [i2t[i] for i in
pred_unk]\\ndf_unk = df_unk[['tweet_id', 'task1', 'ID']]\\
ndf_unk.to_csv(BASE+'Predictions\\\\'+pred_fname, encoding='utf8', index=None)\\
n\""
      ]
     },
     "execution_count": 27,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "'''\n",
    "i2t = ['NOT', 'HOF']\n",
    "df_unk['task1'] = [i2t[i] for i in pred_unk]\n",
    "df_unk = df_unk[['tweet_id', 'task1', 'ID']]\n",
    "df_unk.to_csv(BASE+'Predictions\\\\'+pred_fname, encoding='utf8',
index=None)\n",
    "'''"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.7.6"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```