March 20, 2018

Lab Assignment 4

Sanchit Singhal

INF 385T – Introduction to Machine Learning with Danna Gurari

Spring 2018

School of Information

Hyperlink to code: https://introtoml-sanchit1276.notebooks.azure.com/nb/notebooks/IntroToML/LabAssignment4.ipynb

1. Classification Using Hand-Crafted Features

a) Downloaded dataset

I am only using 101 training samples and 31 validation samples because Jupyter Notebooks is timing out with a data rate error when trying to load the full dataset.

b) Feature Extraction

In [2]:
```
import cv2
import json
import os
import pprint
import requests

import matplotlib.pyplot as plt
import numpy as np

from skimage import io

import nltk
nltk.download('all')

%matplotlib inline
```

```
[nltk_data] Downloading collection 'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to
[nltk_data]    |     C:\Users\sanch\AppData\Roaming\nltk_data...
[nltk_data]    |   Package abc is already up-to-date!
[nltk_data]    | Downloading package alpino to
[nltk_data]    |     C:\Users\sanch\AppData\Roaming\nltk_data...
[nltk_data]    |   Package alpino is already up-to-date!
[nltk_data]    | Downloading package biocreative_ppi to
[nltk_data]    |     C:\Users\sanch\AppData\Roaming\nltk_data...
[nltk_data]    |   Package biocreative_ppi is already up-to-date!
[nltk_data]    | Downloading package brown to
[nltk_data]    |     C:\Users\sanch\AppData\Roaming\nltk_data...
[nltk_data]    |   Package brown is already up-to-date!
[nltk_data]    | Downloading package brown_tei to
[nltk_data]    |     C:\Users\sanch\AppData\Roaming\nltk_data...
[nltk_data]    |   Package brown_tei is already up-to-date!
[nltk_data]    | Downloading package cess_cat to
[nltk_data]    |     C:\Users\sanch\AppData\Roaming\nltk_data...
[nltk_data]    |   Package cess_cat is already up-to-date!
```

```
# set up global variables
subscription_key_cv = "ab0796f0206841acb8c82559e980a458"
subscription_key_ta = "0cd70c24933e4a71acd32335e53d3a23"

vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/"
vision_analyze_url = vision_base_url + "analyze?"

text_analytics_base_url = "https://westcentralus.api.cognitive.microsoft.com/text/analytics/v2.0/"
language_api_url = text_analytics_base_url + "languages"
sentiment_api_url = text_analytics_base_url + "sentiment"
key_phrase_api_url = text_analytics_base_url + "keyPhrases"

image_dir = './'
train_image_dir = './Train/'
test_image_dir = './Validation/'
pred_image_dir = './Test/'
```

```
# evaluate an image
def extract_high_level_features(image_path):
    headers  = {'Ocp-Apim-Subscription-Key': subscription_key_cv,
                'Content-Type'            : 'application/octet-stream'}
    params   = {'visualFeatures': 'Adult,Categories,Description,Color,Faces,ImageType,Tags'}
    body     = open(image_path, 'rb')

    response = requests.post(vision_analyze_url, headers=headers, params=params, data=body)
    response.raise_for_status()
    data = response.json()

    features = {
        "clip_art_type"                    : data["imageType"]["clipArtType"],
        "line_drawing_type"                : data["imageType"]["lineDrawingType"],
        "is_black_and_white"               : int(data["color"]["isBwImg"]),
        "is_adult_content"                 : int(data["adult"]["isAdultContent"]),
        "adult_score"                      : data["adult"]["adultScore"],
        "is_racy"                          : int(data["adult"]["isRacyContent"]),
        "racy_score"                       : data["adult"]["racyScore"],
        "has_faces"                        : int(len(data["faces"])),
        "num_faces"                        : len(data["faces"]),
        "is_dominant_color_background_black" : int(data["color"]["dominantColorBackground"] == "Black"),
        "is_dominant_color_foreground_black" : int(data["color"]["dominantColorForeground"] == "Black")
    }

    return list(features.values())
```

```
# initialize HOG descriptor
width, height = 128, 64
descriptor = cv2.HOGDescriptor(_winSize = (width,height),
                               _blockSize = (16,16),
                               _blockStride = (8,8),
                               _cellSize = (8,8),
                               _nbins = 9)

# compute HOG features for an image
def extract_low_level_features(image_path, descriptor=descriptor):
    image = io.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (width,height))

    hog = descriptor.compute(image)
    hog = hog.flatten().tolist()
    return hog
```

```
# image feature extraction

def extract_all_features(image_dir):
    features = []
    for image_path in os.listdir(image_dir):
        path = image_dir+image_path
        high_level_features = extract_high_level_features(path)
        low_level_features = extract_low_level_features(path)
        features.append(high_level_features + low_level_features)

    return features

image_features_train = extract_all_features(train_image_dir)
image_features_test = extract_all_features(test_image_dir)
image_features_pred = extract_all_features(pred_image_dir)
```

```
# evaluate text "documents"

def analyze_text(documents):
    headers   = {"Ocp-Apim-Subscription-Key": subscription_key_ta}

    # query API for language analysis
    response  = requests.post(language_api_url, headers=headers, json=documents)
    languages = response.json()

    # query API for sentiment analysis
    response  = requests.post(sentiment_api_url, headers=headers, json=documents)
    sentiments = response.json()

    # query API for key phrases extraction
    response  = requests.post(key_phrase_api_url, headers=headers, json=documents)
    key_phrases = response.json()

    return (languages, sentiments, key_phrases)
```

```
# conduct POS tagging and return counts of POS types
def pos_tag_text(documents):
    tag_counts = []
    for document in documents['documents']:
        text = nltk.word_tokenize(document['text'])
        tags = nltk.pos_tag(text)
        frequencies = nltk.FreqDist(tag for (word, tag) in tags)

        tag_count = {
            'ADJ': 0,
            'ADP': 0,
            'ADV': 0,
            'CONJ': 0,
            'DET': 0,
            'NOUN': 0,
            'NUM': 0,
            'PRT': 0,
            'PRON': 0,
            'VERB': 0,
            '.': 0,
            'X': 0
        }

        for tag, count in frequencies.most_common():
            tag_count[tag] = count

        tag_counts.append(tag_count)

    return tag_counts
```

```python
# conduct POS tagging and return counts of POS types
def pos_tag_text(documents):
    tag_counts = []
    for document in documents['documents']:
        text = nltk.word_tokenize(document['text'])
        tags = nltk.pos_tag(text)
        frequencies = nltk.FreqDist(tag for (word, tag) in tags)

        tag_count = {
            'ADJ': 0,
            'ADP': 0,
            'ADV': 0,
            'CONJ': 0,
            'DET': 0,
            'NOUN': 0,
            'NUM': 0,
            'PRT': 0,
            'PRON': 0,
            'VERB': 0,
            '.': 0,
            'X': 0
        }

        for tag, count in frequencies.most_common():
            tag_count[tag] = count

        tag_counts.append(tag_count)

    return tag_counts
```

```python
# get the question documents from the training set
def get_questions(num_of_questions, url):
    labels = json.load(open(url))

    documents = []
    for index, document in enumerate(labels):
        if index == num_of_questions:
            break
        documents.append({
            'id': index,
            'text': document['question']
        })

    return {'documents': documents}
```

```
documents_train = get_questions(100, './train.json')
documents_test = get_questions(31, './val.json')
documents_pred = get_questions(31, './test.json')
```

```
# text feature extraction

languages_train, sentiments_train, key_phrases_train = analyze_text(documents_train)
languages_train, sentiments_train, key_phrases_train = languages_train['documents'], sentiments_train['documents'], key_phrases_t
tag_counts_train = pos_tag_text(documents_train)

languages_test, sentiments_test, key_phrases_test = analyze_text(documents_test)
languages_test, sentiments_test, key_phrases_test = languages_test['documents'], sentiments_test['documents'], key_phrases_test['
tag_counts_test = pos_tag_text(documents_test)

languages_pred, sentiments_pred, key_phrases_pred = analyze_text(documents_test)
languages_pred, sentiments_pred, key_phrases_pred = languages_pred['documents'], sentiments_pred['documents'], key_phrases_pred['
tag_counts_pred = pos_tag_text(documents_pred)
```

```
text_features_train = []
text_features_test = []
text_features_pred = []

for i in range(len(documents_train['documents'])):
    analysis = {
            'sentiment': sentiments_train[i]['score'],
            'numKeyPhrases': len(key_phrases_train[i]['keyPhrases'])
                }
    text_features_train.append({**analysis, **tag_counts_train[i]})


for i in range(len(documents_test['documents'])):
    analysis = {
            'sentiment': sentiments_test[i]['score'],
            'numKeyPhrases': len(key_phrases_test[i]['keyPhrases'])
                }
    text_features_test.append({**analysis, **tag_counts_test[i]})

for i in range(len(documents_pred['documents'])):
    analysis = {
            'sentiment': sentiments_test[i]['score'],
            'numKeyPhrases': len(key_phrases_pred[i]['keyPhrases'])
                }
    text_features_pred.append({**analysis, **tag_counts_pred[i]})
```

```
# get the class labels for the instances
def get_class_labels(url, size):
    train_labels = json.load(open(url))
    train_classes = [label['answerable'] for label in train_labels[:size]]
    return train_classes
```

```
train_classes = get_class_labels('./train.json', 102)
test_classes = get_class_labels('./val.json', 31)
```

c) Training/Validating models for selection

```
import pandas as pd
from pandas import DataFrame
from IPython.display import display

# Transform features and build multi-modal DFs to build models for selection

df_image_features_train = DataFrame(data=image_features_train)
df_image_features_test = DataFrame(data=image_features_test)
df_image_features_pred = DataFrame(data=image_features_pred)

df_text_features_train = DataFrame(data=text_features_train)
df_text_features_test = DataFrame(data=text_features_test)
df_text_features_pred = DataFrame(data=text_features_pred)

df_train_mm = pd.concat([df_image_features_train, df_text_features_train],axis=1)
df_test_mm = pd.concat([df_image_features_test, df_text_features_test],axis=1)
df_pred_mm = pd.concat([df_image_features_pred, df_text_features_pred],axis=1)

# clean and reshape

df_train_mm['name'] = 'train'
df_test_mm['name'] = 'test'
df_pred_mm['name'] = 'pred'

concat_df = pd.concat([df_train_mm , df_test_mm,df_pred_mm], axis=0, ignore_index=True)
concat_df = concat_df.fillna(0)

df_train_mm = concat_df[concat_df['name'] == 'train']
df_test_mm = concat_df[concat_df['name'] == 'test']
df_pred_mm = concat_df[concat_df['name'] == 'pred']

df_train_mm = df_train_mm.drop('name', axis=1)
df_test_mm = df_test_mm.drop('name', axis=1)
df_pred_mm = df_pred_mm.drop('name', axis=1)
```

```
from sklearn.metrics import accuracy_score

# Evaluating Naive Bayes

from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X=df_train_mm, y=train_classes)
classes_nb = nb.predict(df_test_mm)

print(classes_nb)
print(accuracy_score(classes_nb, test_classes))

# Evaluating Logistic Regression

from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(X=df_train_mm, y=train_classes)
classes_logreg = logreg.predict(df_test_mm)

print(classes_logreg)
print(accuracy_score(classes_logreg, test_classes))

# Evaluating Decision Tree

from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
dt.fit(X=df_train_mm, y=train_classes)
classes_dt = dt.predict(df_test_mm)

print(classes_dt)
print(accuracy_score(classes_dt, test_classes))

# Evaluating Random Forest

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
rf.fit(X=df_train_mm, y=train_classes)
classes_rf = rf.predict(df_test_mm)

print(classes_rf)
print(accuracy_score(classes_rf, test_classes))
```

```
[1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 1 1 0 1 0 1 1 0 0 1 1 1 1 0 1 0]
0.451612903226
[1 1 1 1 1 0 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1]
0.58064516129
[1 0 0 1 0 0 1 1 1 1 0 0 0 1 1 0 1 0 1 1 0 1 0 0 0 1 1 1 0 1 1]
0.548387096774
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1]
0.741935483871
```

d) Test Predictions with chosen method

```
# Predictions for the test set using selected classification system

classes_final_predictions = rf.predict(df_pred_mm)

print(classes_final_predictions)
```

```
[1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0]
```

e) Proposed Prediction Method

My proposed prediction method was the Random Forest Classifier. Initial analysis of the datasets showed me that there was a lot of data and I struggled even loading the entire set into my notebook with the computing resource available to me. Immediately I knew that I would not be able to train my model with all this data and therefore I would need to select a model that does not absolutely require a large training set. At the start, I thought a decision tree would perform the best – which it did but then I realized that ensemble learning could be used to enhance this model to a Random Forest. Further the Random Forest is a predictive tool rather than a descriptive tool – this works in my favor as we are only interested in predicting the classes.

Just to confirm this hunch, I built several models using the training set (of only 101 examples) and tested the performance on the validation set (only 31 examples) – Random Forest did indeed performed the best. This makes sense to me as the random forest can be flexible and able to incorporate different kinds of features.

Design decisions were based on the complexity of the underlying relationship. Building a multi-modal that uses computer vision and natural language processing to concatenate input features and then predict a class is a complex problem. On top of that, the task of predicting if a visual question is answerable is clearly highly non-linear and therefore I decided to go the decision tree/ random forest route – non-linear relationship tend not to affect its performance as much.

It would be interesting for me to run a similar analysis but using the complete dataset (given enough time and resources). The Random Forest takes a lot of time to train and therefore I would need a longer time/faster computer. I suspect that my classification system is quite rudimentary as it is trained with such a small amount of data and I would be curious to see how the other models perform when built with a larger training set.

2. Classification Using Neural Networks

a. Load MNIST and create 70/30 train/test split

In [1]:
```
# 2a
```

In [2]:
```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
```

In [3]:
```
print(mnist.keys())
```

```
dict_keys(['DESCR', 'COL_NAMES', 'target', 'data'])
```

In [4]:
```
print(mnist.data.shape)
print(mnist.target.shape)
```

```
(70000, 784)
(70000,)
```

In [5]:
```
from sklearn.model_selection import train_test_split

X_train_mnist, X_test_mnist, y_train_mnist, y_test_mnist = train_test_split(mnist.data, mnist.target, test_size=0.3, random_state
```

In [11]:
```
print(X_train_mnist.shape)
print(y_train_mnist.shape)
print(X_test_mnist.shape)
print(y_test_mnist.shape)
```

```
(49000, 784)
(49000,)
(21000, 784)
(21000,)
```

b. Optimize Hyperparameters – 10 hidden layers and 10 number of neurons

```
from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()
stdsc.fit(X_train_mnist)
X_train_mnist__std = stdsc.transform(X_train_mnist)
X_test_mnist_std = stdsc.transform(X_test_mnist)
```

```
from sklearn.neural_network import MLPClassifier
import numpy as np

n_hidden_nodes = list(range(1, 11))
n_hidden_layers = list(range(1, 11))

nodes = []
accuracy = []
layers = []


counter = [1]
counter = np.array(counter)

for l in n_hidden_layers:

    for n in n_hidden_nodes:

        mlp = MLPClassifier(activation='tanh', hidden_layer_sizes=counter)
        mlp.fit(X_train_mnist__std, y_train_mnist)
        acc = mlp.score(X_test_mnist_std, y_test_mnist)

        print(counter)
        counter = counter + 1

        nodes.append(n)
        layers.append(l)
        accuracy.append(acc)

    counter[counter > .5] = 1
    counter = np.append(counter,1)
```

```
print(accuracy[accuracy.index(max(accuracy))])
print(nodes[accuracy.index(max(accuracy))])
print(layers[accuracy.index(max(accuracy))])
```

```
0.916952380952
10
5
```

c. Optimal Hyperparameters and number of weights found

As seen from the results above, the optimal hyperparameters were 10 neurons per layer and 5 hidden layers. This neural network gave me an accuracy of 91.7%. Below is my calculations for the number of weights and parameters:

# of inputs = 784, # of outputs = 1, 10 neurons per layer, 5 hidden layers

# of weights = (784*10)+(10x10)+(10x10)+(10x10))+(10x10)+(10*1) = 8250

# of parameters = 8250 weights + 784 inputs = 9034

d. Discussion regarding performance of neural network with different hyperparameters

The performance of the neural network differs depending the hyperparameters used. When considering the number of nodes independent of the numbers of layers used, it is clear from the scatter plot below that the highest accuracy occurs when the number of neurons is at 10 per layer. There is a definite positive correlation between the number of nodes in a layer and the performance of the network. Although, the incremental gain after around 6 nodes per layer is not as much as going from 1 neuron to 5 neurons.

On the other hand, the number of hidden layers did not seem to contribute much to the performance of the network. Increasing or decreasing the number of hidden layers had no affect on the accuracy as the performance was poor for a low number of nodes at both 2 layers and 10 layers. Similarly, the performance was good at 10 nodes at both 2 layers and 10 layers.

These results make sense to me as increasing the number of neurons per layers should allow for a more intricate fit of the data – although I suppose, increasing the number of neurons to thousands can lead to overfitting – and that is why the accuracy is increasing when number of nodes is largest in this case. In contrast, it also makes sense to me that increasing the number of hidden layers does not help the accuracy. Adding more number of hidden layers enables the modelling of highly non-linear relationships and the fact that adding layers does not affect performance tells me that the underlying function in the data – it the relationship was more non-linear, I would expect the number of hidden layers to have a greater affect.