U Intern Tasks

Part1:

Q1:

using cumsum function in numpy library to calculate the number with multiplication in 10.

Q2:

Using time.clock() function at before starting of program and after finishing it ,then subtract start time from end time in u second.

Q4:

Making for loop count frequency of text and return char frequency.

Q5:

Making function with if statement to change word as required.

Q6:

from datetime compute the current time and using timedelta to add 5 seconds.

Q7:

Using equation of polynomial in numpy.

Q8:

Using equation of determinant of an array in numpy.

Part2

Creating cal.py using Button of Tkinter with different color like the required image.

Fake and real news classification

Introduction:

We have 2 datasets with Real News data and Fake News data. Each dataset has around 22000 articles. As we merge both, we have around 45000 articles of real and fake. The aim is to build a model to correctly predict if a news is real or fake.

Dependencies include python libraries like:

1-sklearn

2-pandas

3-matpltlib

4-Numpy

5- nltk

Algorithms used:

1-Decition Tree

2-Random forest

3-Logistic Regression

4-Niave Bayes

5- K Nearest

6- Gradient Boosting

7- XGBC

Steps of project:

1-Data Preprocessing:

Preprocessing Text to get Stemmed and Lemmatized Corpus:

1. Removing all the extra information like brackets, any kind of punctuations - commas, apostrophes, quotes, question marks, and more.
2. Remove all the numeric text, urls
3. Remove all the stop words like - just, and, or, but, if
4. Convert all the remaining text in the lower case, separated by space
5. Generate stemmed text
6. Generate lemmatized text

2-Classification models using CountVectorizer and TFIDF Vectorizer:

classification models in 2 steps-

1. Using stemmed text and both Count Vectorizer and TFIDF Vectorizer
2. Using lemmatized text and both Count Vectorizer and TFIDF Vectorizer

3-Results:

After doing Predictive analysis above using both stemmed text and lemmatized text with Count Vectorizer and TFIDF Vectorizer, we see that the XGB Classifier is giving best results with lemmatized texts and Count Vectorizer with accuracy score of 99.78%

MNIST data classification

Introduction

MNIST ("Modified National Institute of Standards and Technology") is the de facto “hello world” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. A new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike. In this competition, your goal is to correctly identify digits from a dataset of tens of thousands of handwritten images. We’ve curated a set of tutorial-style kernels which cover everything from regression to neural networks. We encourage you to experiment with different algorithms to learn first-hand what works well and how techniques compare.

Data Description

The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero).

For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

Visually, if we omit the "pixel" prefix, the pixels make up the image like this:

```
000 001 002 003 ... 026 027
```

```
028 029 030 031 ... 054 055

056 057 058 059 ... 082 083

 |   |   |   |  ...  |   |

728 729 730 731 ... 754 755

756 757 758 759 ... 782 783
```

The test data set, (test.csv), is the same as the training set, except that it does not contain the "label" column.

Your submission file should be in the following format: For each of the 28000 images in the test set, output a single line containing the ImageId and the digit you predict. For example, if you predict that the first image is of a 3, the second image is of a 7, and the third image is of a 8, then your submission file would look like:

```
ImageId,Label
1,3
2,7
3,8

(27997 more lines)
```

Steps of project:

1- **Data preparation:**

1) perform a grayscale normalization to reduce the effect of illumination's differences.

Moreover the CNN converg faster on [0..1] data than on [0..255].

2) Reshape image in 3 dimensions (height = 28px, width = 28px.

3) Label encoding using One-Hot Encoding:

Encode labels to one hot vectors (like: 2 -> [0,0,1,0,0,0,0,0,0,0])

4) Split the train and the validation set for the fitting:

split the train set in two parts : a small fraction (10%) became the validation set which the model is evaluated and the rest (90%) is used to train the model.

5)Apply standardization by using mean and standard deviation.

2-Creating CNN Model:

1)Define the model:

-Using Keras Sequential API, where you have just to add one layer at a time, starting from the input.

-The first is the convolutional (Conv2D) layer. It is like a set of learnable filters. setting 32 filters for the two firsts conv2D layers and 64 filters for the two second layers and 128 filters for two third layers and 256 for the last ones. Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image.

-The CNN can isolate features that are useful everywhere from these transformed images (feature maps).

-The second important layer in CNN is the pooling (MaxPool2D) layer. This layer simply acts as a downsampling filter. It looks at the 2 neighboring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. We have to choose the pooling size (i.e the area size pooled each time) more the pooling dimension is high, more the downsampling is important.

-Combining convolutional and pooling layers, CNN are able to combine local features and learn more global features of the image.

-'relu' is the rectifier (activation function max(0,x). The rectifier activation function is used to add non linearity to the network.

-The Flatten layer is use to convert the final feature maps into a one single 1D vector. This flattening step is needed so that you can make use of fully connected layers after some convolutional/maxpool layers. It combines all the found local features of the previous convolutional layers.

-In two fully-connected (Dense) layers which is just artificial an neural networks (ANN) classifier. In the last layer(Dense(10,activation="softmax")) the net outputs distribution of probability of each class.

2- Data augmentation:

In order to avoid overfitting problem, we need to expand artificially our handwritten digit dataset. We can make your existing dataset even larger. The idea is to alter the training data with small transformations to reproduce the variations occurring when someone is writing a digit.

-Without data augmentation i obtained an accuracy of 98.114%

-With data augmentation i achieved 99.67% of accuracy

3) Model training:

Fit the model with 50 epochs and show accuracy of each epoch.

4) Confusion matrix:

Confusion matrix can be very helpful to see your model drawbacks.

3- Prediction validation results:

Finally, I applied prediction function on data to show results of real number and predict number.