Deep Learning: Language identification using Keras & TensorFlow

April 29, 2017 (http://croock.webfactional.com/deep-learning-language-identification-using-keras-tensorflow/) Lucas KM (http://croock.webfactional.com/author/lukas_km/) Projects (http://croock.webfactional.com/category/projects/)

Welcome to my second Data Science project. This time we will dive into the most recent & hot technology: Deep Neural Networks (DNN).

The problem I am going tackle here is the following: can we identify the language of short text (140 characters) with high accuracy using neural networks? This problem is currently solved by various software libraries, but using a set of hardcoded rules and lookup tables. We will attack this problem using Machine Learning algorithms.

The (Imaginary) Intro Story

The intro story is fictional, and is here to help you unrderstand potential business context of the problem. What we are doing here is Applied Machine Learning, after all! 🙂

You are a new employee in R&D department of ACME Inc., a large IT corporation. You have been employed due to your skills in Deep Learning alogorithms.

ACME, within its countless departments, has also a NLPS (Natural Language Processing Services) department. This department offers many services for language translation, sentiment analysis, text classifiation. Among these services, they offer also language identification service. The service is working considerably well, offering accuracy at 95% level. The NLPS executive director believes this level can be much higher. If ACME could claim they have language identification accuracy at 99% level, they could gain huge advantage on the market and market its services as being the best language identification services in the world.

Therefore, you are asked to build a prototype language identification solution using Deep Learning approach. If you prove this solution can achieve 95% accuracy or more, ACME's NLPS director will invest into full scale research and impementation project with you as its leader.

The Solution

In order to solve this problem, I decided to build a Deep Neural Network. In order to find the best solution for this problem, I tested several approaches:

A) First approach was based on feeding the Neural Network with text sample that was processed in the following way:

- first, words were sorted alphabetcally (this is quite important - without this step accuracy dropped significantly)
- then, all chars were encoded into Unicode numbers (using Python ord() function).

That approach was working quite OK, achieving accuracy around 89%. But that was not enough for me.

B) Second approach was to add other processed information to data from approach A. So I have added:

- a few most popular substrings in words (like "ed" is a common English substring in words like worked, readed etc)
- a list of most popular letters
- a list of non-ASCII letters in the string
- again, all letters have been encoded to Unicode

That approach worked even better, achieving accuracy around 93%. But that was still not enough for me. While experimenting with this approach and looking for more ways to improve accuracy I decided to test another approach:

C) I decided to count occurence frequency of all the possible letters in a sample text string. Assuming we have 7 languages, we exactly know which letters can occur in such texts. List of such letters is just sum of alphabets of all these languages, without letters repetitions. And for each letter in such alphabet we can count its ocurrence in a sample text.

And that was it! Frankly, I expected that this approach could give me some clues about the text's language, but the result was amazing:

The letter frequency approach in language identification using neural networks can achieve 97,56% accuracy!

This result is just for 7 languages, but some of them are quite similiar to each other. Italian, Spanish and French are considered to be in Latin language group, English and German have also common roots. Czech and Slovakian are extremely similar and are considered to be one of major challenged in the language recognition (see https://en.wikipedia.org/wiki/Language_identification)

Please see the solution details below and run the code yourself.

The Notebook

Please review the notebook below for details of the solution.

You can also reviev the Notebook on Github (https://github.com/lukaszkm/machinelearningexp/blob/master/Deep_Learning_Language_identification.ipynb).

00. Goal

Your R&D task is the following:

1. Select a few languages that use similiar alphabet (between 5 and 10 languages)
2. Among these languages, there should be at least 2 that are

considered to be very similiar to each other (and therefore challenging for language identification)
3. Build a prototype of Deep Learning solution that will have more than 95% accuracy in language identification (classification) task
4. The accuracy should be checked based on reference texts of length 140 characters (Tweet/SMS length)
5. Other classifiction parameters (like precision and recall) should be well balanced

I decided to recognize language of text using Deep Neural Network. For this, I will build a prototype DNN classifier for 7 European languages: English, German, French, Italian, Spanish, Czech and Slovakian. We will train and test it with texts sourced from Wikipedia.
NOTE: This notebook is CPU and memory consuming. You have been warned 🙂

01. Notebook Intro & Imports

Deep Learning Language Identification Notebook

- Notebook @author Lukasz Kamieniecki-Mruk, lucas.mlexp@gmail.com, http://machinelearningexp.com (http://machinelearningexp.com)
- Notebook License: Creative Commons CC-BY-SA https://creativecommons.org/licenses/by-sa/4.0/ (https://creativecommons.org/licenses/by-sa/4.0/)
- Dataset source: https://dumps.wikimedia.org (https://dumps.wikimedia.org)
- Dataset license: GNU Free Documentation License (GFDL) and the Creative Commons Attribution-Share-Alike 3.0 License

In [2]:

```
# imports
import os
import re
import math
import random
import collections
import time
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import classification_report
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
import keras.optimizers
from keras.utils import plot_model
print("Keras backend : ", keras.backend.backend())
```

```
Using TensorFlow backend.
```

```
Keras backend :  tensorflow
```

In [3]:

```
# key variables
# dictionary of languages that our classifier will cover
languages_dict = {'en':0,'fr':1,'es':2,'it':3,'de':4,'sk':
5,'cs':6}
# length of cleaned text used for training and prediction
- 140 chars
text_sample_size = 140
# number of language samples per language that we will ext
ract from source files
num_lang_samples = 250000

# utility function to turn language id into language code
def decode_langid(langid):
    for dname, did in languages_dict.items():
```

```
        if did == langid:
            return dname

# utility function to return file Bytes size in MB
def size_mb(size):
    size_mb =  '{:.2f}'.format(size/(1000*1000.0))
    return size_mb + " MB"

# we will use alphabet for text cleaning and letter counti
ng
def define_alphabet():
    base_en = 'abcdefghijklmnopqrstuvwxyz'
    special_chars = ' !?¿¡'
    german = 'äöüß'
    italian = 'àèéìíòóùú'
    french = 'àâæçéèêêîïôœùûüÿ'
    spanish = 'áéíóúüñ'
    czech = 'áčďéěíjňóřšťúůýž'
    slovak = 'áäčďdzdžéíĺľňóôŕšťúýž'
    all_lang_chars = base_en + german +  italian + french
+ spanish + czech + slovak
    small_chars = list(set(list(all_lang_chars)))
    small_chars.sort()
    big_chars = list(set(list(all_lang_chars.upper())))
    big_chars.sort()
    small_chars += special_chars
    letters_string = ''
    letters = small_chars + big_chars
    for letter in letters:
        letters_string += letter
    return small_chars,big_chars,letters_string

alphabet = define_alphabet()
print (alphabet)
```

```
(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', '
l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
'x', 'y', 'z', 'ß', 'à', 'á', 'â', 'ä', 'æ', 'ç', 'è', 'é'
, 'ê', 'ì', 'í', 'î', 'ï', 'ñ', 'ò', 'ó', 'ô', 'ö', 'ù', '
ú', 'û', 'ü', 'ý', 'ÿ', 'č', 'ď', 'ě', 'ĺ', 'ľ', 'ň', 'œ',
'ŕ', 'ř', 'š', 'ť', 'ů', 'ž', ' ', '!', '?', '¿', '¡'], ['
A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X'
, 'Y', 'Z', 'À', 'Á', 'Â', 'Ä', 'Æ', 'Ç', 'È', 'É', 'Ê', '
Ì', 'Í', 'Î', 'Ï', 'Ñ', 'Ò', 'Ó', 'Ô', 'Ö', 'Ù', 'Ú', 'Û',
'Ü', 'Ý', 'Č', 'Ď', 'Ě', 'Ĺ', 'Ľ', 'Ň', 'Œ', 'Ŕ', 'Ř', 'Š'
, 'Ť', 'Ů', 'Ÿ', 'Ž'], 'abcdefghijklmnopqrstuvwxyzßàáâäæçè
éêìíîïñòóôöùúûüýÿčďěĺľňœŕřšťůž !?¿¡ABCDEFGHIJKLMNOPQRSTUVW
XYZÀÁÂÄÆÇÈÉÊÌÍÎÏÑÒÓÔÖÙÚÛÜÝČĎĚĹĽŇŒŔŘŠŤŮŸŽ')
```

02. Raw Data preparation

Before I could run this notebook, I had to prepare the data.
What we need is a big block of text for each language we want to identify. We will sample randomly from each block during our DNN training, validation and test.

[A]. I downloaded Wikipedia dump files from page: https://dumps.wikimedia.org (https://dumps.wikimedia.org).
I decided to download " All pages, current versions only" and files packed into bz2 archives, less compressed than 7zip files, thus smaller after decompression. Example file name: enwiki-20170301-pages-meta-current1.xml-p000000010p000030303.bz2

I downloaded random files for each of the following languages:

- English (enwiki files)
- French (frwiki files)
- Spanish (eswiki files)
- Italian (itwiki files)
- German (dewiki files)
- Czech (cswiki files)
- Slovakian (skwiki files)

My package had around 8,5 GB of data so prepare enough storage on your disk.

[B]. I extracted text from the files using Wikipedia Extractor Python package http://medialab.di.unipi.it/wiki/Wikipedia_Extractor (http://medialab.di.unipi.it/wiki/Wikipedia_Extractor)
I run the extractor with parameters to get files of around 100 MB in size. This way, I could easily select some of them to create data for our Neural Network.
Consult Wikipedia Extractor page for details of using this package.
Example command in shell: python WikiExtractor.py -b 100000K /users/luke/Documents/ML\ Datasets/WIKI_DUMPS/enwiki-20170301-pages-meta-current1.xml-p000000010p000030303.bz2

[C]. As a result, I got several extracted files for each language (named wiki_00, wiki_01 etc):

- I selected manually some of them (pseudo-randomly 😉 to get approximately 200 MB of data for each language.
- I merged the files for each language to get one, big file for each language. I did merging with my Mac shell "cat" command. You can use command from your OS or use python to do that.
- I named the files according to 2-digit language ISO code. As a result, I have tteh following files in my data directory now:
  en.txt, fr.txt, es.txt, it.txt, de.txt, cs.txt, sk.txt

In [4]:

```
# I keep raw data in 'original' subfolder, and cleaned dat
a in 'cleaned' subfolder
# 'samples' subdirectory is for files with text samples pr
ocessed according to my sampling procedure
# 'train_test' subdirecrory is for files with np.arrays pr
epared for NN train and test data (both features and targe
ts)
data_directory = "./Data/"
source_directory = data_directory + 'source'
cleaned_directory = data_directory + 'cleaned'
samples_directory = data_directory + 'samples'
train_test_directory = data_directory + 'train_test'

for filename in os.listdir(source_directory):
    path = os.path.join(source_directory, filename)
    if not filename.startswith('.'):
        print((path), "size : ",size_mb(os.path.getsize(pa
th)))
```

```
./Data/source/cs.txt size :  204.79 MB
./Data/source/de.txt size :  204.76 MB
./Data/source/en.txt size :  204.75 MB
./Data/source/es.txt size :  204.77 MB
./Data/source/fr.txt size :  204.72 MB
./Data/source/it.txt size :  204.78 MB
./Data/source/sk.txt size :  204.78 MB
```

In [5]:

```
# we will create here several text-cleaning procedures.
# These procedure will help us to clean the data we have f
or training,
# but also will be useful in cleaning the text we want to
classify, before the classification by trained DNN

# remove XML tags procedure
# for example, Wikipedia Extractor creates tags like this
below, we need to remove them
# <doc id="12" url="https://en.wikipedia.org/wiki?curid=12
" title="Anarchism"> ... </doc>
def remove_xml(text):
    return re.sub(r'<[^<]+?>', '', text)

# remove new lines - we need dense data
def remove_newlines(text):
    return text.replace('\n', ' ')

# replace many spaces in text with one space - too many sp
aces is unnecesary
# we want to keep single spaces between words
# as this can tell DNN about average length of the word an
d this may be useful feature
def remove_manyspaces(text):
    return re.sub(r'\s+', ' ', text)

# and here the whole procedure together
def clean_text(text):
    text = remove_xml(text)
    text = remove_newlines(text)
    text = remove_manyspaces(text)
    return text
```

In [6]:

```
for lang_code in languages_dict:
    path_src = os.path.join(source_directory, lang_code+".
txt")
    f = open(path_src)
    content = f.read()
    print('Language : ',lang_code)
    print ('Content before cleaning :-> ',content[1000:100
0+text_sample_size])
    f.close()
    # cleaning
    content = clean_text(content)
    print ('Content after cleaning :-> ',content[1000:1000
+text_sample_size])
    path_cl = os.path.join(cleaned_directory,lang_code + '
_cleaned.txt')
    f = open(path_cl,'w')
    f.write(content)
    f.close()
    del content
    print ("Cleaning completed for : " + path_src,'->',pat
h_cl)
    print (100*'-')
print ("END OF CLEANING")
```

```
Language :  en
Content before cleaning :->  osophy. Many types and tradit
ions of anarchism exist, not all of which are mutually exc
lusive. Anarchist schools of thought can differ funda
Content after cleaning :->  lly exclusive. Anarchist schoo
ls of thought can differ fundamentally, supporting anythin
g from extreme individualism to complete collectivis
Cleaning completed for : ./Data/source/en.txt -> ./Data/cl
eaned/en_cleaned.txt
----------------------------------------------------------
----------------------------------------
Language :  fr
Content before cleaning :->   très tôt ses distances avec
l'esprit contestataire de 1968.

Avec plus de quatorze prix et neuf nominations, l'art ciné
matographique de Pie
Content after cleaning :->  f nominations, l'art cinématog
raphique de Pier Paolo Pasolini s'impose, dès 1962 avec no
tamment "L'Évangile selon saint Matthieu", puis avec
```

```
Cleaning completed for : ./Data/source/fr.txt -> ./Data/cl
eaned/fr_cleaned.txt
----------------------------------------------------------
----------------------------------------
Language :  es
Content before cleaning :->  a_4, y formula_5, formula_6 s
on las integrales elípticas de primera y segunda especie.

Una ecuación aproximada de su superficie es:
donde p
Content after cleaning :->  a ecuación aproximada de su su
perficie es: donde p ≈ 1,6075. Con esta expresión se obtie
ne un error máximo de ±1,061%, en función de los val
Cleaning completed for : ./Data/source/es.txt -> ./Data/cl
eaned/es_cleaned.txt
----------------------------------------------------------
----------------------------------------
Language :  it
Content before cleaning :->  erie A2.

Nella stagione 2003-04, oltre alla vittoria della Coppa It
alia di Serie A2, la formazione veronese, vince il campion
ato, senza per
Content after cleaning :->  azione veronese, vince il camp
ionato, senza perdere neppure una delle trenta partita del
la "regular season", ottenendo la promozione in mass
Cleaning completed for : ./Data/source/it.txt -> ./Data/cl
eaned/it_cleaned.txt
----------------------------------------------------------
----------------------------------------
Language :  de
Content before cleaning :->  nenmarkts und die Vereinheitl
ichung fiskalisch-ökonomischer Rahmenbedingungen. Politisc
h stärkte der Deutsche Zollverein die Vormachtstellun
Content after cleaning :->   der Deutsche Zollverein die V
ormachtstellung Preußens und förderte die Entstehung der k
leindeutschen Lösung. Nach der Gründung des Deutsche
Cleaning completed for : ./Data/source/de.txt -> ./Data/cl
eaned/de_cleaned.txt
----------------------------------------------------------
----------------------------------------
Language :  sk
Content before cleaning :->  speranto využíva pri cestovan
í, korešpondencii, medzinárodných stretnutiach a kultúrnyc
h výmenách, kongresoch, vedeckých diskusiách, v pôvod
```

```
Content after cleaning :->  , divadle a kine, hudbe, tlače
nom aj internetovom spravodajstve, rozhlasovom a televízno
m vysielaní. Slovná zásoba esperanta pochádza predov
Cleaning completed for : ./Data/source/sk.txt -> ./Data/cl
eaned/sk_cleaned.txt
----------------------------------------------------------
------------------------------------------
Language :  cs
Content before cleaning :->  rd a absolutní výškový rekord
. Letouny tvořily podstatnou část amerických leteckých sil
i v 70. a 80. letech 20. století, kdy začaly být nah
Content after cleaning :->  80. letech 20. století, kdy za
čaly být nahrazovány modernějšími stroji, jako například F
-15 a F-16 u amerického letectva, F-14 a F/A-18 u am
Cleaning completed for : ./Data/source/cs.txt -> ./Data/cl
eaned/cs_cleaned.txt
----------------------------------------------------------
------------------------------------------
END OF CLEANING
```

03. Input Data Preparation

In [7]:

```
# this function will get sample of texh from each cleaned
language file.
# It will try to preserve complete words - if word is to b
e sliced, sample will be shortened to full word
def get_sample_text(file_content,start_index,sample_size):
    # we want to start from full first word
    # if the firts character is not space, move to next on
es
    while not (file_content[start_index].isspace()):
        start_index += 1
    #now we look for first non-space character - beginning
of any word
    while file_content[start_index].isspace():
        start_index += 1
    end_index = start_index+sample_size
    # we also want full words at the end
    while not (file_content[end_index].isspace()):
        end_index -= 1
    return file_content[start_index:end_index]

# we need only alpha characters and some (very limited) sp
```

```
ecial characters
# exactly the ones defined in the alphabet
# no numbers, most of special characters also bring no val
ue for our classification task
# (like dot or comma - they are the same in all of our lan
guages so does not bring additional informational value)

# count number of chars in text based on given alphabet
def count_chars(text,alphabet):
    alphabet_counts = []
    for letter in alphabet:
        count = text.count(letter)
        alphabet_counts.append(count)
    return alphabet_counts

# process text and return sample input row for DNN
# note that we are counting separatey:
# a) counts of all letters regardless of their size (whole
text turned to lowercase letter)
# b) counts of big letters only
# this is because German uses big letters for beginning of
nouns so this feature is meaningful
def get_input_row(content,start_index,sample_size):
    sample_text = get_sample_text(content,start_index,samp
le_size)
    counted_chars_all = count_chars(sample_text.lower(),al
phabet[0])
    counted_chars_big = count_chars(sample_text,alphabet[1
])
    all_parts = counted_chars_all + counted_chars_big
    return all_parts

# let's see if our processing is returning counts
# last part calculates also input_size for DNN so this cod
e must be run before DNN is trained
path = os.path.join(cleaned_directory, "de_cleaned.txt")
with open(path, 'r') as f:
    content = f.read()
    random_index = random.randrange(0,len(content)-2*text_
sample_size)
    sample_text = get_sample_text(content,random_index,tex
t_sample_size)
    print ("1. Sample text: \n",sample_text)
    print ("2. Reference alphabet: \n",alphabet[0],alphabe
t[1])
```

```
    sample_input_row = get_input_row(content,random_index,
text_sample_size)
    print ("3. Sample_input_row: \n",sample_input_row)
    input_size = len(sample_input_row)
    print ("4. Input size : ", input_size)
    del content
```

```
1. Sample text:
 der Existenz der so bezeichneten Eisenbahngesellschaft be
kannt sein, um die Eigentümerschaft und damit auch die Bed
eutung der Eisenbahn aus
2. Reference alphabet:
 ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', '
l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
'x', 'y', 'z', 'ß', 'à', 'á', 'â', 'ä', 'æ', 'ç', 'è', 'é'
, 'ê', 'ì', 'í', 'î', 'ï', 'ñ', 'ò', 'ó', 'ô', 'ö', 'ù', '
ú', 'û', 'ü', 'ý', 'ÿ', 'č', 'ď', 'ě', 'ĺ', 'ľ', 'ň', 'œ',
'ŕ', 'ř', 'š', 'ť', 'ů', 'ž', ' ', '!', '?', '¿', '¡'] ['A
', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X'
, 'Y', 'Z', 'À', 'Á', 'Â', 'Ä', 'Æ', 'Ç', 'È', 'É', 'Ê', '
Ì', 'Í', 'Î', 'Ï', 'Ñ', 'Ò', 'Ó', 'Ô', 'Ö', 'Ù', 'Ú', 'Û',
'Ü', 'Ý', 'Č', 'Ď', 'Ě', 'Ĺ', 'Ľ', 'Ň', 'Œ', 'Ŕ', 'Ř', 'Š'
, 'Ť', 'Ů', 'Ÿ', 'Ž']
3. Sample_input_row:
 [8, 5, 4, 8, 24, 2, 3, 6, 9, 0, 1, 2, 3, 13, 1, 0, 0, 4,
9, 8, 6, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 18, 0, 0, 0, 0, 0, 1, 0, 0, 4, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4. Input size :  132
```

In [8]:

```
# now we have preprocessing utility functions ready. Let's
use them to process each cleaned language file
# and turn text data into numerical data samples for our n
eural network
# prepare numpy array
sample_data = np.empty((num_lang_samples*len(languages_dic
t),input_size+1),dtype = np.uint16)
lang_seq = 0
jump_reduce = 0.2 # part of characters removed from jump t
```

```
o avoid passing the end of file
for lang_code in languages_dict:
    start_index = 0
    path = os.path.join(cleaned_directory, lang_code+"_cle
aned.txt")
    with open(path, 'r') as f:
        print ("Processing file : " + path)
        file_content = f.read()
        content_length = len(file_content)
        remaining = content_length - text_sample_size*num_
lang_samples
        jump = int(((remaining/num_lang_samples)*3)/4)
        print ("File size : ",size_mb(content_length),\
               " | # possible samples : ",int(content_leng
th/input_size),\
              "| # skip chars : " + str(jump))
        for idx in range(num_lang_samples):
            input_row = get_input_row(file_content,start_i
ndex,text_sample_size)
            sample_data[num_lang_samples*lang_seq+idx,] =
input_row + [languages_dict[lang_code]]
            start_index += text_sample_size + jump
        del file_content
    lang_seq += 1
    print (100*"-")

# let's randomy shuffle the data
np.random.shuffle(sample_data)
# reference input size
print ("Input size : ",input_size )
print (100*"-")
print ("Samples array size : ",sample_data.shape )
path_smpl = os.path.join(samples_directory,"lang_samples_"
+str(input_size)+".npz")
np.savez_compressed(path_smpl,data=sample_data)
print(path_smpl, "size : ",size_mb(os.path.getsize(path_sm
pl)))
del sample_data
```

```
Processing file : ./Data/cleaned/en_cleaned.txt
File size :  198.62 MB  | # possible samples :  1504721 |
# skip chars : 490
-----------------------------------------------------------
---------------------------------------
Processing file : ./Data/cleaned/fr_cleaned.txt
File size :  193.28 MB  | # possible samples :  1464219 |
# skip chars : 474
-----------------------------------------------------------
---------------------------------------
Processing file : ./Data/cleaned/es_cleaned.txt
File size :  196.82 MB  | # possible samples :  1491081 |
# skip chars : 485
-----------------------------------------------------------
---------------------------------------
Processing file : ./Data/cleaned/it_cleaned.txt
File size :  194.70 MB  | # possible samples :  1474979 |
# skip chars : 479
-----------------------------------------------------------
---------------------------------------
Processing file : ./Data/cleaned/de_cleaned.txt
File size :  196.80 MB  | # possible samples :  1490940 |
# skip chars : 485
-----------------------------------------------------------
---------------------------------------
Processing file : ./Data/cleaned/sk_cleaned.txt
File size :  170.60 MB  | # possible samples :  1292458 |
# skip chars : 406
-----------------------------------------------------------
---------------------------------------
Processing file : ./Data/cleaned/cs_cleaned.txt
File size :  174.49 MB  | # possible samples :  1321868 |
# skip chars : 418
-----------------------------------------------------------
---------------------------------------
Input size :  132
-----------------------------------------------------------
---------------------------------------
Samples array size :  (1750000, 133)
./Data/samples/lang_samples_132.npz size :  56.98 MB
```

In [10]:

```
# now we will review the data  - control check step
path_smpl = os.path.join(samples_directory,"lang_samples_"
+str(input_size)+".npz")
dt = np.load(path_smpl)['data']
random_index = random.randrange(0,dt.shape[0])
print ("Sample record : \n",dt[random_index,])
print ("Sample language : ",decode_langid(dt[random_index,
][input_size]))
# we can also check if the data have equal share of differ
ent languages
print ("Dataset shape :", dt.shape)
bins = np.bincount(dt[:,input_size])
print ("Language bins count : ")
for lang_code in languages_dict:
    print (lang_code,bins[languages_dict[lang_code]])
```

```
Sample record :
 [12  0  0  2 10  0  2  2 11  1  5  8  4  2  3  1  0  3  9
7  2  2  3  0  1
  0  0  0  2  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0
0  0  0  0  0  0
  0  1  0  0  0  0  0  0  0  0  1  0  0  0 24  0  0  0  0
0  0  0  0  0  0
  0  0  6  0  2  2  0  0  0  0  0  0  0  2  0  0  1  0  0
0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0
  0  0  0  0  0  0  0  5]
Sample language :  sk
Dataset shape : (1750000, 133)
Language bins count :
en 250000
fr 250000
es 250000
it 250000
de 250000
sk 250000
cs 250000
```

In [11]:

```
# we need to preprocess data for DNN yet again - scale it
# scling will ensure that our optimization algorithm (vari
ation of gradient descent) will converge well
```

```
# we need also ensure one-hot econding of target classes f
or softmax output layer
# let's convert datatype before processing to float
dt = dt.astype(np.float64)
# X and Y split
X = dt[:,0:input_size]
Y = dt[:,input_size]
del dt
# random index to check random sample
random_index = random.randrange(0,X.shape[0])
print("Example data before processing:")
print("X : \n", X[random_index,])
print("Y : \n", Y[random_index])
time.sleep(120) # sleep time to allow release memory. This
step is very memory consuming
# X preprocessing
# standar scaler will be useful laterm during DNN predicti
on
standard_scaler = preprocessing.StandardScaler().fit(X)
X = standard_scaler.transform(X)
print ("X preprocessed shape :", X.shape)
# Y one-hot encoding
Y = keras.utils.to_categorical(Y, num_classes=len(language
s_dict))
# See the sample data
print("Example data after processing:")
print("X : \n", X[random_index,])
print("Y : \n", Y[random_index])
# train/test split. Static seed to have comparable results
for different runs
seed = 42
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.20, random_state=seed)
del X, Y
# wait for memory release again
time.sleep(120)
# save train/test arrays to file
path_tt = os.path.join(train_test_directory,"train_test_da
ta_"+str(input_size)+".npz")
np.savez_compressed(path_tt,X_train=X_train,Y_train=Y_trai
n,X_test=X_test,Y_test=Y_test)
print(path_tt, "size : ",size_mb(os.path.getsize(path_tt))
)
del X_train,Y_train,X_test,Y_test
```

```
Example data before processing:
X :
 [ 15.   2.   6.   5.  13.   1.   2.   0.  11.   0.   0.
8.   1.   7.   6.
   3.   3.   4.   5.   2.   6.   2.   0.   0.   0.   0.
0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   1.   0.   0.   1.
0.   1.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.   0.   0.   0.
   0.   0.   0.   0.  24.   0.   0.   0.   0.   0.   0.
2.   0.   1.   0.
   0.   0.   5.   0.   0.   0.   0.   0.   0.   0.   0.
0.   0.   0.   0.
   2.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.   0.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
0.]
Y :
 2.0
X preprocessed shape : (1750000, 132)
Example data after processing:
X :
 [  1.43848896e+00   2.43704418e-01   1.08847496e+00   1.8
2666132e-01
   1.45227519e-02  -1.80809384e-01   2.47627426e-01  -1.08
427356e+00
   8.66345449e-01  -6.18827831e-01  -7.34121439e-01   9.37
334909e-01
  -1.11405375e+00  -3.61732266e-01  -5.11423692e-01   1.48
601106e-01
   4.05548333e+00  -1.07418909e+00  -5.49680116e-01  -1.59
587561e+00
   9.36575304e-01  -1.46580321e-01  -5.15809788e-01  -3.72
479618e-01
  -7.76121131e-01  -7.73038291e-01  -1.44181107e-01  -2.73
753470e-01
  -5.22001321e-01  -7.93495188e-02  -2.63451748e-01  -1.93
269396e-02
  -9.38347781e-02  -2.74018573e-01  -5.75990888e-01  -1.26
350120e-01
  -8.11002284e-02   2.45440751e-01  -7.58357766e-02  -4.54
601720e-02
```

```
   5.12772045e+00  -1.28507660e-01   1.59196714e+00  -1.66
344824e-01
  -1.92302478e-01  -1.20958020e-01  -3.12401117e-01  -4.85
314714e-02
  -2.47004806e-01  -3.91328291e-01  -6.08162563e-03  -3.88
669017e-01
  -1.27498724e-01  -2.99588064e-01  -4.65800157e-02  -2.02
632413e-01
  -1.45005865e-01  -5.37390197e-02  -2.90206661e-02  -2.84
958102e-01
  -3.55911578e-01  -2.02015881e-01  -2.13457830e-01  -3.74
036288e-01
   1.32442461e+00  -4.19795108e-02  -3.83158070e-02  -1.22
594952e-02
  -1.12017023e-02  -4.84387533e-01  -4.19931991e-01   2.80
642142e+00
  -4.17778482e-01   1.54904416e+00  -3.41660767e-01  -3.38
574938e-01
  -3.28867911e-01   8.14616426e+00  -3.08094259e-01  -3.16
254182e-01
  -4.43012420e-01  -4.41307729e-01  -3.59292660e-01  -2.83
874098e-01
  -4.63330589e-01  -1.11120589e-01  -3.55856685e-01  -5.33
646163e-01
  -4.14336686e-01  -2.50622432e-01   3.90821724e+00  -2.57
584435e-01
  -9.72753900e-02  -1.12086879e-01  -2.23296209e-01  -5.63
501467e-02
  -4.19133185e-02  -1.77236964e-02  -3.59801525e-02  -1.15
182751e-02
  -9.39162050e-03  -4.82739747e-02  -8.04572674e-02  -4.59
818488e-03
  -1.42857289e-03  -2.09963245e-02  -1.76056377e-02  -1.51
185962e-03
  -6.03612665e-03  -2.13809482e-03  -2.01344765e-02  -4.55
847042e-03
  -4.25821128e-02  -1.51185962e-03  -5.70288459e-02  -1.51
185962e-03
  -5.18277097e-02  -6.11770067e-03  -1.15180032e-01  -3.39
250404e-02
  -4.42762723e-03  -1.30930846e-03  -2.96007223e-02  -4.10
516922e-03
  -7.78726852e-03  -7.55929162e-04  -4.47537672e-02  -9.20
400841e-02
  -1.58758260e-02  -2.13809482e-03  -7.55929162e-04  -6.51
```

```
882518e-02]
Y :
 [ 0.  0.  1.  0.  0.  0.  0.]
./Data/train_test/train_test_data_132.npz size :  147.32 M
B
```

04. Train & Test Deep Neural Network

In [12]:

```
# load train data first from file
path_tt = os.path.join(train_test_directory,"train_test_da
ta_"+str(input_size)+".npz")
train_test_data = np.load(path_tt)
X_train = train_test_data['X_train']
print ("X_train: ",X_train.shape)
Y_train = train_test_data['Y_train']
print ("Y_train: ",Y_train.shape)
X_test = train_test_data['X_test']
print ("X_test: ",X_test.shape)
Y_test = train_test_data['Y_test']
print ("Y_test: ",Y_test.shape)
del train_test_data
```

```
X_train:  (1400000, 132)
Y_train:  (1400000, 7)
X_test:  (350000, 132)
Y_test:  (350000, 7)
```

In [13]:

```
# create DNN using Keras Sequential API
# I added Dropout to prevent overfitting
model = Sequential()
model.add(Dense(500,input_dim=input_size,kernel_initialize
r="glorot_uniform",activation="sigmoid"))
model.add(Dropout(0.5))
model.add(Dense(300,kernel_initializer="glorot_uniform",ac
tivation="sigmoid"))
model.add(Dropout(0.5))
model.add(Dense(100,kernel_initializer="glorot_uniform",ac
tivation="sigmoid"))
model.add(Dropout(0.5))
model.add(Dense(len(languages_dict),kernel_initializer="gl
orot_uniform",activation="softmax"))
model_optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0
.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
model.compile(loss='categorical_crossentropy',
              optimizer=model_optimizer,
              metrics=['accuracy'])
```

In [14]:

```
# let's fit the data
# history variable will help us to plot results later
history = model.fit(X_train,Y_train,
          epochs=12,
          validation_split=0.10,
          batch_size=64,
          verbose=2,
          shuffle=True)
```

```
Train on 1260000 samples, validate on 140000 samples
Epoch 1/12
316s - loss: 0.1345 - acc: 0.9582 - val_loss: 0.0841 - val
_acc: 0.9723
Epoch 2/12
265s - loss: 0.1020 - acc: 0.9691 - val_loss: 0.0806 - val
_acc: 0.9736
Epoch 3/12
275s - loss: 0.0976 - acc: 0.9704 - val_loss: 0.0787 - val
_acc: 0.9743
Epoch 4/12
246s - loss: 0.0946 - acc: 0.9710 - val_loss: 0.0773 - val
_acc: 0.9749
Epoch 5/12
349s - loss: 0.0930 - acc: 0.9717 - val_loss: 0.0771 - val
_acc: 0.9753
Epoch 6/12
249s - loss: 0.0921 - acc: 0.9721 - val_loss: 0.0768 - val
_acc: 0.9750
Epoch 7/12
273s - loss: 0.0911 - acc: 0.9725 - val_loss: 0.0757 - val
_acc: 0.9757
Epoch 8/12
270s - loss: 0.0904 - acc: 0.9727 - val_loss: 0.0750 - val
_acc: 0.9759
Epoch 9/12
250s - loss: 0.0901 - acc: 0.9729 - val_loss: 0.0753 - val
_acc: 0.9765
Epoch 10/12
247s - loss: 0.0892 - acc: 0.9729 - val_loss: 0.0748 - val
_acc: 0.9761
Epoch 11/12
216s - loss: 0.0891 - acc: 0.9732 - val_loss: 0.0748 - val
_acc: 0.9764
Epoch 12/12
208s - loss: 0.0890 - acc: 0.9733 - val_loss: 0.0743 - val
_acc: 0.9762
```

In [15]:

```
# now we will face the TRUTH. What is our model real accur
acy tested on unseen data?
scores = model.evaluate(X_test, Y_test, verbose=1)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*10
0))
```

```
349824/350000 [============================>.] - ETA: 0s
acc: 97.56%
```

In [16]:

```
# and now we will prepare data for scikit-learn classifica
tion report
Y_pred = model.predict_classes(X_test)
Y_pred = keras.utils.to_categorical(Y_pred, num_classes=le
n(languages_dict))
```

```
349696/350000 [============================>.] - ETA: 0s
```

In [17]:

```
# and run the report
target_names =  list(languages_dict.keys())
print(classification_report(Y_test, Y_pred, target_names=t
arget_names))
```

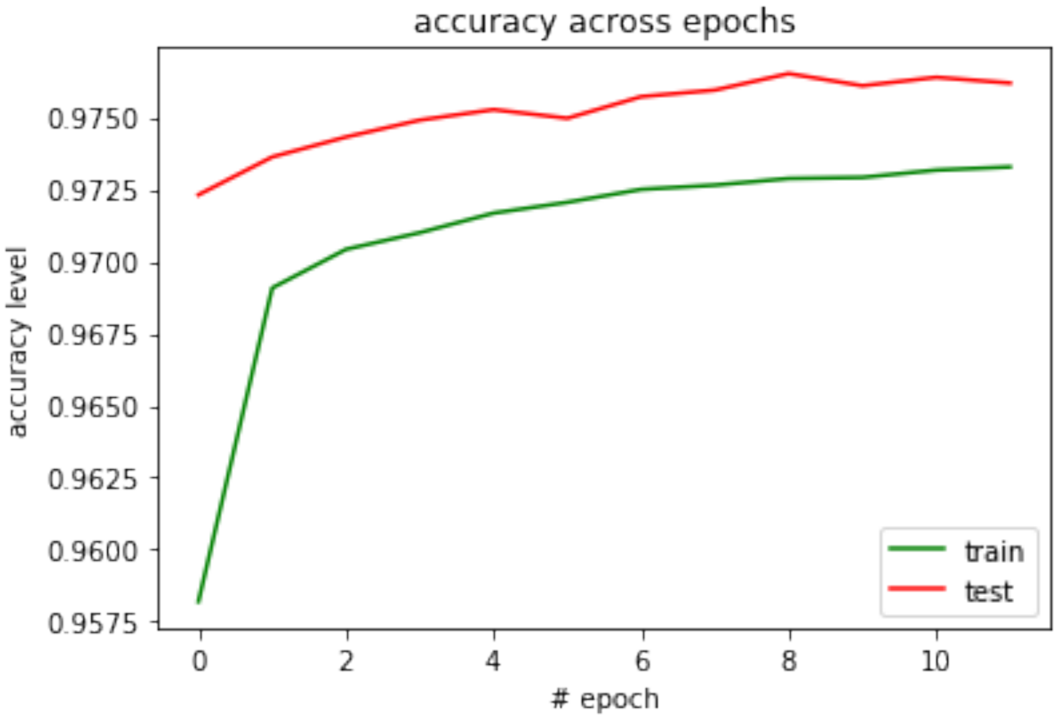
```
             precision    recall  f1-score   support

         en       0.95      0.98      0.97     50285
         fr       0.98      0.98      0.98     50003
         es       0.98      0.98      0.98     50206
         it       0.97      0.96      0.97     49599
         de       0.99      0.98      0.99     50024
         sk       0.97      0.98      0.97     49812
         cs       0.98      0.97      0.98     50071

avg / total       0.98      0.98      0.98    350000
```

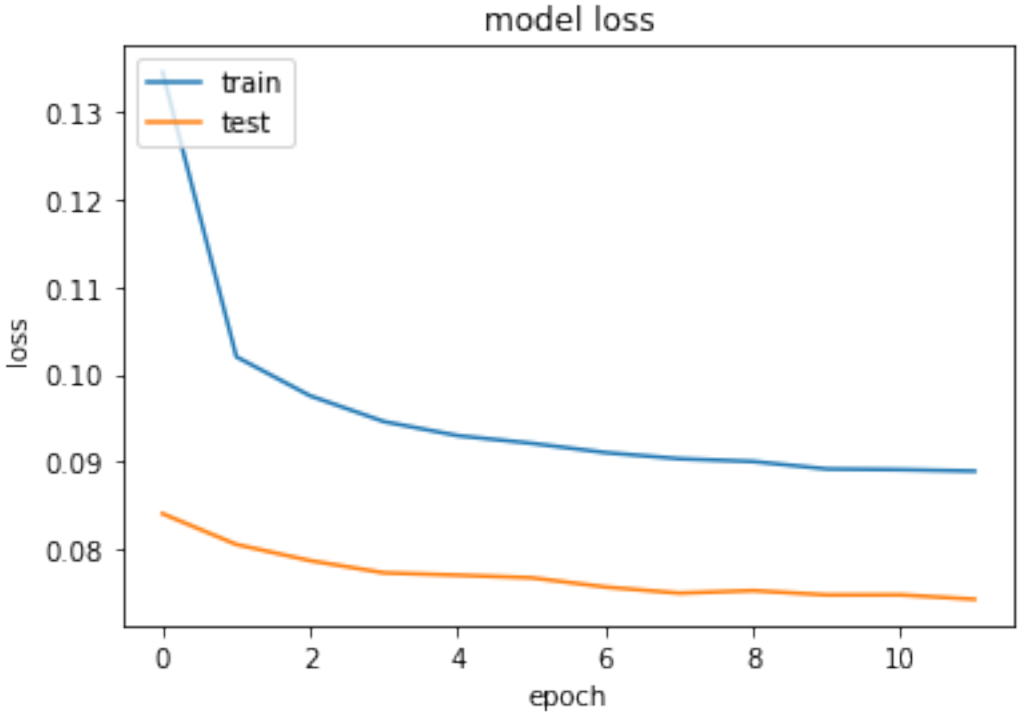
In [18]:

```
# show plot accuracy changes during training
plt.plot(history.history['acc'],'g')
plt.plot(history.history['val_acc'],'r')
plt.title('accuracy across epochs')
plt.ylabel('accuracy level')
plt.xlabel('# epoch')
plt.legend(['train', 'test'], loc='lower right')
plt.show()
```



In [19]:

```
# show plot of loss changes during training
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [20]:

```
# and now we will have some fun. Seeing is believing!
# We will take some texts and try to predict the text's la
nguage using our trained neural network.

# Frank Baum, The Wonderful Wizard of Oz, Project Gutenber
g, public domain
en_text = "You are welcome, most noble Sorceress, to the l
and of the Munchkins. We are so grateful to you \
for having killed the Wicked Witch of the East, and for se
tting our people free from bondage."
# Johann Wolfgang von Goethe, Faust: Der Tragödie erster T
eil, Project Gutenberg, public domain
de_text = "Habe nun, ach! Philosophie, Juristerei und Medi
zin, Und leider auch Theologie \
Durchaus studiert, mit heißem Bemühn. Da steh ich nun, ich
armer Tor! Und bin so klug als wie zuvor;"
# Pierre Benoît, L'Atlantide,
fr_text = "Voilà cinq mois que j'en faisais fonction, et,
ma foi, je supportais bien cette responsabilité et \
goûtais fort cette indépendance. Je puis même affirmer, sa
ns me flatter"
# Alberto Boccardi, Il peccato di Loreta, Project Gutenber
g, public domain
it_text = "Giovanni Sant'Angelo, che negli anni passati a
Padova in mezzo alla baraonda tanto gioconda degli student
i,\
aveva appreso ad amare con foga di giovane qualche alto id
eale, tornato in famiglia dovette fare uno sforzo"
```

```
# Fernando Callejo Ferrer, Música y Músicos Portorriqueños
, Project Gutenberg, public domain
es_text = "Dedicada esta sección a la reseña de los compos
itores nativos y obras que han producido, con ligeros \
comentarios propios a cada uno, parécenos oportuno dar lig
eras noticias sobre el origen de la composición"
# František Omelka, Blesky nad Beskydami, Project Gutenber
g, public domain
cs_text = "A Slávek, jsa povzbuzen, se ptal a otec odpovíd
al. Přestože byl prostým venkovským listonošem,\
nepřivedla jej žádná synova otázka do rozpaků. Od mládí se
zajímal o dějepis a literaturu."
# Janko Matúška, Nad Tatrou sa blýska,  national anthem of
Slovakia, https://en.wikipedia.org/wiki/Nad_Tatrou_sa_blýs
ka
sk_text = "Nad Tatrou1 sa blýska Hromy divo bijú Zastavme
ich, bratia Veď sa ony stratia Slováci ožijú \
To Slovensko naše Posiaľ tvrdo spalo Ale blesky hromu Vzbu
dzujú ho k tomu Aby sa prebralo"

text_texts_array = [en_text,de_text,fr_text,it_text,es_tex
t,cs_text,sk_text]
test_array = []
for item in text_texts_array:
    cleaned_text = clean_text(item)
    input_row = get_input_row(cleaned_text,0,text_sample_s
ize)
    test_array.append(input_row)

test_array = standard_scaler.transform(test_array)
Y_pred = model.predict_classes(test_array)
for id in range(len(test_array)):
    print ("Text:",text_texts_array[id][:50],"... -> Predi
cted lang: ", decode_langid(Y_pred[id]))
```

```
7/7 [==============================] - 0s
Text: You are welcome, most noble Sorceress, to the land .
.. -> Predicted lang:  en
Text: Habe nun, ach! Philosophie, Juristerei und Medizin .
.. -> Predicted lang:  de
Text: Voilà cinq mois que j'en faisais fonction, et, ma  .
.. -> Predicted lang:  fr
Text: Giovanni Sant'Angelo, che negli anni passati a Pad .
.. -> Predicted lang:  it
Text: Dedicada esta sección a la reseña de los composito .
.. -> Predicted lang:  es
Text: A Slávek, jsa povzbuzen, se ptal a otec odpovídal. .
.. -> Predicted lang:  cs
Text: Nad Tatrou1 sa blýska Hromy divo bijú Zastavme ich .
.. -> Predicted lang:  sk
```

05. Summary

Let’s check whether we have achieved our goals:

1. Select a few languages that use similiar alphabet (between 5 and 10 languages)
    - PASSED, we have 7 languages in total. All are used for training and validation of the model
2. Among these languages, there should be at least 2 that are considered to be very similiar to each other (and therefore challenging for language identification)
    - PASSED, we have Czech and Slovakian, a very similiar languages. Both nations had a common state and share hundreds of years of common history and culture.
3. Build a prototype of Deep Learning solution that will have more than 95% accuracy in language identification (classification) task
    - PASSED, our accuracy is nearly 98% (97.92% on test data)
4. The accuracy should be checked based on reference texts of length 140 characters (Tweet/SMS length)
    - PASSED, our sample texts are 140 characters long
5. Other classifiction parameters (like precision and recall) should be well balanced
    - PASSED, average precision is 98% and recall is also 98% (both

values have maximum deviation no bigger than 1%)

Related

Deep Learning: Predicting hard disks failures using recurrent LSTM network (http://croock.webfactional.com/deep-learning-predicting-hard-disks-failures-using-recurrent-lstm-network/)
July 12, 2018
In "Projects"

Machine Learning: Regression of 911 Calls (http://croock.webfactional.com/machine-learning-regression-911-calls/)
February 24, 2017
In "Projects"

Data Science: Performance of Python vs Pandas vs Numpy (http://croock.webfactional.com/data-science-performance-of-python-vs-pandas-vs-numpy/)
July 15, 2017
In "Tips and Tricks"

classification (http://croock.webfactional.com/tag/classification/)

Deep Neural Networks (http://croock.webfactional.com/tag/deep-neural-networks/)

DNN (http://croock.webfactional.com/tag/dnn/)

Keras (http://croock.webfactional.com/tag/keras/)

Neural Networks (http://croock.webfactional.com/tag/neural-networks/)

NLP (http://croock.webfactional.com/tag/nlp/)

Python (http://croock.webfactional.com/tag/python/)

Scikit-Learn (http://croock.webfactional.com/tag/scikit-learn/)

TensorFlow (http://croock.webfactional.com/tag/tensorflow/)

LUCAS KM (HTTP://CROOCK.WEBFACTIONAL.COM/AUTHOR/LUKAS_KM/)

My name is Lucas and I am an Senior IT Business Anayst with over 10 years of experience in IT system development and integration. I have worked for several large international companies from industries such as: IT, finance, insurance, telecommunications, pharmacy and gambling. I took part in multiple IT projects, ranging from small budgets (hundreds thousands of US dollars) to large (multiple millions of US dollars). A year ago (in 2016), encouraged by a friend, I completed Andrew Ng Machine Learning course on Coursera and I got fascinated by this subject. So I decided to master it. Feel free to contact me via my Email address: lucas.mlexp@gmail.com / Linkedin profile: https://www.linkedin.com/in/lukaszmruk / Twitter account: https://twitter.com/ml_exp

‹ Machine Learning: Regression of 911 Calls (http://croock.webfactional.com/machine-learning-regression-911-calls/)

Data Science: Performance of Python vs Pandas vs Numpy ›
(http://croock.webfactional.com/data-science-performance-of-python-vs-pandas-vs-

numpy/)