# Italian Dialects: NLP For Local Linguistics

The idea behind this project is the task proposed by GeoLingIt Shared Task and published on Avalita, in which, given a dataset containing tweets written in Italian dialect associated with the region of origin of the dialect, you had to predict the region of origin of a dialect text never seen. This project extends the task with an extra phase: the translation of the dialect text in italian.

```
#installation of the necessary libraries
!pip install cleantext
!pip install spacy
!pip install keras
!pip install nltk
!pip install -U spaCy
!python -m spacy download it_core_news_sm
!pip install tensorflow
```

Mostra output nascosti

## *Analysis of the dataset*

The starting dataset consists of two parts: dev and train. Analyzing the number of sentences associated with each Italian region, we noticed a remarkable imbalance and a small number of examples. As a result the two files were merged and a over-sampling phase was planned (detailed view later). Analyzing the sentences, we saw that they needed a preprocessing phase to normalize everything and leave only the text we needed.

```
#Import of the necessary libraries
import pandas as pd
import spacy
import re
from tqdm import tqdm
from cleantext import clean
import numpy as np
from random import randint
```

```
#Loading of the train dataset
df = None
with open('TRAIN_NLP_DIALECT.csv', 'r', encoding='latin-1') as f:
    for i,line in enumerate(f.readlines()):
        if i == 0:
          columns = line.strip().split(';')
          columns = columns[1:]
          df = pd.DataFrame(columns=columns)
        else:
          line = line.lower()
          row = line.strip().split(';')[1:]
          if len(row) > len(columns):
            target = row.pop(-1)
            val = ''
            for i, el in enumerate(row):
              val += el
              if i < len(row)-1:
                val += ';'
            row = [val]
            row.append(target)
          df.loc[i] = row
```

```
#Loading of the dev dataset
dev = None
with open('FinalTest.csv', 'r', encoding='latin-1') as f:
    for i,line in enumerate(f.readlines()):
        if i == 0:
          columns = line.strip().split(',')
          columns = columns[2:]
          dev = pd.DataFrame(columns=columns)
        else:
          line = line.lower()
          row = line.strip().split(',')[2:]
          if len(row) > len(columns):
            target = row.pop(-1)
            val = ''
            for i, el in enumerate(row):
              val += el
              if i < len(row)-1:
                val += ','
```

```
            row = [val]
            row.append(target)
            dev.loc[len(dev)] = row
```

```
print(dev)
```

```
                                                  text     region
0     mortacci, na roba che nse po' vede, por, na c...      lazio
1     ou belin, ma mi avevano detto che non finiva ...    liguria
2     ora che sta a casa da due anni, a capit ca ni...   campania
3     e er boja stava all'ordine der giorno. adesso...      lazio
4     quando e uscito 50 sfumature di grigio, tutte...   calabria
..                                                 ...        ...
183   distratto. no. ieri no gho vu tempo e anco so...     veneto
184   belin coerenza, sono riusciti in 2anni ad ann...    liguria
185   incredibilmente, alla lunga, ne sono usciti b...  lombardia
186   che domenica e senza : so maista' u cannolu s...    sicilia
187   quando decideva di giocarla sul serio, ce n'e...     puglia

[188 rows x 2 columns]
```

```
df['region'].value_counts()
```

```
region
lazio                    5587
campania                 3016
veneto                    764
lombardia                 688
sicilia                   612
toscana                   418
sardegna                  359
emilia romagna            319
calabria                  281
puglia                    264
piemonte                  236
liguria                   223
friuli-venezia giulia     218
marche                    179
abruzzo                   150
umbria                    136
trentino-alto adige        52
basilicata                 49
molise                     35
valle d'aosta              14
Name: count, dtype: int64
```

```
dev['region'].value_counts()
```

```
region
campania                 30
lazio                    27
lombardia                21
emilia romagna           17
toscana                  16
veneto                   15
sicilia                  11
liguria                   9
friuli-venezia giulia     9
puglia                    9
calabria                  8
sardegna                  8
piemonte                  8
Name: count, dtype: int64
```

```
df['region'] = df['region'].str.lower()
dev['region'] = dev['region'].str.lower()
```

Here we can see the unbalance of the dataset and the presence of Minonitary classes.

## Pre processing

The initial datasets were in tsv format to allow different users to work with different libraries on them. Since pandas was used in this project, the dataset was converted to csv format and this led to the automatic addition of the column 'Unnamed: 0' that, in this preprocessing sentence, we will remove.

```
ds = df.drop(['Unnamed: 0'],axis = 1)
dev = dev.drop(['Unnamed: 0'], axis = 1)
```

Now the actual preprocessing phase begins. Then let’s remove from the phrases: Twitter tags (current X), emoticons and hashtags.

```
#Definition of the text cleaning function
def clean_the_text(text: str):
  pattern = r'\[.*?\]|\#\w+'
  cleaned_text = re.sub(pattern, '', text)
  cleaned_text = clean(cleaned_text, no_emoji=True)
  return cleaned_text
```

We apply the function to the two datasets:

```
#Pre-processing of the train dataset
ids = ds['id'].to_numpy()
new_text = []

for id in tqdm(ids):
  clean_text = clean_the_text(ds[ds['id']==id]['text'].values[0])
  new_text.append(clean_text)

ds['text'] = new_text
ds
```

```
#Pre-processing of the dev dataset
ids = dev['id'].to_numpy()
new_text = []

for id in tqdm(ids):
  clean_text = clean_the_text(dev[dev['id']==id]['text'].values[0])
  new_text.append(clean_text)

dev['text'] = new_text
dev
```

Then we proceed with the union of the two datasets, also to have a greater number of total examples.

```
ds = pd.concat([df,dev], ignore_index=True)
```

```
ds['region'].value_counts()
```

```
region
lazio                    5614
campania                 3046
veneto                    779
lombardia                 709
sicilia                   623
toscana                   434
sardegna                  367
emilia romagna            336
calabria                  289
puglia                    273
piemonte                  244
liguria                   232
friuli-venezia giulia     227
marche                    179
abruzzo                   150
umbria                    136
trentino-alto adige        52
basilicata                 49
molise                     35
valle d'aosta              14
Name: count, dtype: int64
```

```
ds.to_csv('NLP_Dataset.csv')
```

```
df
```

| | text | region |
|---|---|---|
| 0 | Sò dispiacente ca nun m'ha datu tempu de prepa... | marche |
| 1 | Tornarò a Ascoli a festa de Pasca,. | marche |
| 2 | A me m'ha detto ca t'aspettava a jesi,. | marche |
| 3 | La gùrdia a stava a guardà,. | marche |
| 4 | Porca muntagna si iva a cadè,. | marche |
| ... | ... | ... |
| 14720 | distratto. no. ieri no gho vu tempo e anco so... | veneto |
| 14721 | belin coerenza, sono riusciti in 2anni ad ann... | liguria |
| 14722 | incredibilmente, alla lunga, ne sono usciti b... | lombardia |
| 14723 | che domenica e senza : so maista' u cannolu s... | sicilia |
| 14724 | quando decideva di giocarla sul serio, ce n'e... | puglia |

14725 rows × 2 columns

The dataset we will work on will be as follows:

```
df = pd.read_csv("NLP_Dataset.csv")
```

It has 14725 examples, but the Minonitary classes always remain.

## Over-sampling: selection of the suitable model

To make over sampling we need a model that allow us to generate sentences in italian dialect. So, in this section, different models were tried.

The quality of generative models has been evaluated based on how they generated sentences in Apulian dialect, which presents a regular number of examples on which the model can be based to generate others. Moreover, the Apulian dialect was chosen because we can have a direct evaluation of what was generated. If a model generates a good result for the Apulian dialect, then it will also be used for the dialects of other regions.

### 1. N-GRAM Model

The first model tested is the N-GRAM model. A language model is a probabilistic model that is used to assign a probability to a sequence of words. For example, if we have a group of words and we take the first word, the model can predict the next word, which is the one with the greatest probability of standing next to the first.

```
#Import of the necessary libraries
import nltk
nltk.download("all")

from nltk import word_tokenize
from nltk.lm import MLE
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.tokenize.treebank import TreebankWordDetokenizer
```

Mostra output nascosti

```
#selection of examples from the region of Puglia
puglia = df[df['region']=='puglia']['text']
```

Before applying the model, we must apply the tokenization technique on the dataset examples, that is, divide the phrases into tokens, into pieces. To do this, we use the nltk tokenizer that will output the tokenized text

```
sents = []
for i in puglia:
  s = nltk.sent_tokenize(i)
  sents.append(s)

tokenized_text = [list(map(str.lower, word_tokenize(str(sent))))
                  for sent in sents]
```

Now we can apply the model which, in our case will be a 3-gram model, that is, to calculate the probability of the next word, will consider the above three.

```
n = 3
training_ngrams, padded_sents = padded_everygram_pipeline(n, tokenized_text)
#using a model based on Maximum Likelihood Estimation
model = MLE(n)
model.fit(training_ngrams, padded_sents)

#object we need to take the Tokenized phrase and convert it into a single sentence.
detokenize = TreebankWordDetokenizer().detokenize

#function for generation of sentences
def generate_sent(model, num_words, random_seed):
    content = []
    for token in model.generate(num_words, random_seed=random_seed):
        if token == '<s>':
            continue
        if token == '</s>':
            break
        content.append(token)
    return detokenize(content)

print (generate_sent(model, 15, random_seed=6))
print (generate_sent(model, 15, random_seed=2))
```

```
nan u send canta no pa tutt'appost solo che stavo in fase depressione da campovolo
tieni a mente, lu mare c' e mho a ci non fatica doi
```

The 3-gram model seems to generate valid examples, but, after careful analysis, it has been seen that in reality, the tokenization phase has not been done well. The tokenizer used a basic English dictionary, therefore it does not see every token as a word, but the tokens turn out to be whole sentences. As a result, the same model was tested with a Spacy tokenizer based on an Italian dictionary.

```
#Import of the necessary libraries
from spacy.lang.it import Italian
import spacy
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.tokenize.treebank import TreebankWordDetokenizer

import string

nlp_it = spacy.load("it_core_news_sm")
punctuations = string.punctuation
stop_words_it = spacy.lang.it.stop_words.STOP_WORDS
parser_it = Italian()


# Tokenizer function
def spacy_tokenizer_it(sentence):
    mytokens = parser_it(sentence)
    mytokens = [ word.text for word in mytokens ]
    #removing stop words
    mytokens = [ word for word in mytokens if word not in stop_words_it and word not in punctuations]
    return mytokens


puglia = df[df['region']=='puglia']['text']


sents = []
for i in puglia:
  s = spacy_tokenizer_it(i)
  sents.append(s)

tokenized_text = [list(map(str.lower, word_tokenize(str(sent))))
                  for sent in sents]
```

```
n = 3
training_ngrams, padded_sents = padded_everygram_pipeline(n, tokenized_text)
model = MLE(n)
model.fit(training_ngrams, padded_sents)

detokenize = TreebankWordDetokenizer().detokenize

def generate_sent(model, num_words, random_seed):
    content = []
    for token in model.generate(num_words, random_seed=random_seed):
        if token == '<s>':
            continue
        if token == '</s>':
            break
        content.append(token)
    return detokenize(content)

print (generate_sent(model, 50, random_seed=50))
print (generate_sent(model, 15, random_seed=2))
```

From here we can see that the tokenizer works very well, but the model is not for us because it simply creates a sequence of words and not meaningful sentences.

## 2. Neural networks with Keras

Let's try more complex models based on neural networks made with keras.

### LSTM

A **Long Short-Term Memory (LSTM)** is a type of recurring neural network (RNN) designed to model long-term data sequences. It is particularly useful for natural language processing (NLP) applications such as text generation. LSTM overcomes the fading gradient problem of traditional RNN due to its special architecture that includes memory cells and port mechanisms (input, output and forget) that control the flow of information.

```
puglia = df[df['region']=='puglia']['text']
```

```
#library import
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Embedding, LSTM, Dense, Dropout
from keras.src.models import Sequential
from keras.preprocessing.text import Tokenizer
import keras.src.utils as ku

import warnings
warnings.filterwarnings("ignore")
```

```
from spacy.lang.it import Italian
import string

nlp_it = spacy.load("it_core_news_sm")
punctuations = string.punctuation
stop_words_it = spacy.lang.it.stop_words.STOP_WORDS
parser_it = Italian()
```

```
# Tokenizer function
def spacy_tokenizer_it(sentence):
    mytokens = parser_it(sentence)
    mytokens = [ word.text for word in mytokens ]
    # remove stop words
    mytokens = [ word for word in mytokens if word not in stop_words_it and word not in punctuations ]
    # return preprocessed list of tokens
    return mytokens
```

Neural network-based models need to represent data as token sequences. Accordingly, we define a function to define them.

```
def get_sequence_of_tokens(corpus):
    word_index = {}
    index = 1
    input_sequences = []

    for line in corpus:
        token_list = spacy_tokenizer_it(line)
        token_indices = []
        for token in token_list:
            if token not in word_index:
                word_index[token] = index
                index += 1
            token_indices.append(word_index[token])

        for i in range(1, len(token_indices)):
            n_gram_sequence = token_indices[:i+1]
            input_sequences.append(n_gram_sequence)

    total_words = len(word_index) + 1
    return input_sequences, total_words

inp_sequences, total_words = get_sequence_of_tokens(puglia)
print("Sequence: ",inp_sequences[:10])
print("Total words: ",total_words)
```

```
Sequence:  [[1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2, 3, 4, 5], [1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6, 7], [1, 2, 3, 4, 5, 6, 7, 8],
Total words:  1792
```

However, token sequences can be of variable length, so we define a function to add padding to each sequence to make them the same length.

```
def generate_padded_sequences(input_sequences):
    max_sequence_len = max([len(x) for x in input_sequences])
    input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

    predictors, label = input_sequences[:,:-1],input_sequences[:,-1]
    label = ku.to_categorical(label, num_classes=total_words)
    return predictors, label, max_sequence_len

predictors, label, max_sequence_len = generate_padded_sequences(inp_sequences)
print("Predictors shape:", predictors.shape)
print("Label shape:", label.shape)
print("Max sequence length:", max_sequence_len)
```

```
Predictors shape: (2635, 35)
Label shape: (2635, 1792)
Max sequence length: 36
```

```
def create_model(max_sequence_len, total_words):
    input_len = max_sequence_len - 1
    model = Sequential()

    #add Input Embedding Layer, for internal representation of sequences
    model.add(Embedding(total_words, 20, input_length=input_len))

    #add Hidden LSTM Layer
    model.add(LSTM(200))
    model.add(Dropout(0.2))

    # Add Output Layer
    model.add(Dense(total_words, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam')

    return model

model = create_model(max_sequence_len, total_words)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 35, 20)            35840

 lstm (LSTM)                 (None, 200)               176800

 dropout (Dropout)           (None, 200)               0

 dense (Dense)               (None, 1792)              360192

=================================================================
```

```
Total params: 572832 (2.19 MB)
Trainable params: 572832 (2.19 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.fit(predictors, label, epochs = 3, verbose=1)
```

```
tokenizer = Tokenizer()

def generate_text(seed_text, next_words, model, max_sequence_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
        predicted = model.predict(token_list, verbose=0)
        predicted = np.argmax(predicted, axis=-1)

        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text.title()
```

```
seed_text = "Lu"
next_words = 5
generated_text = generate_text(seed_text, next_words, model, max_sequence_len)
print(generated_text)
```

```
Lu
```

This model does not work for our goal.

## VAE

Now create a VAE network for text generation from scratch. A **Variational Autoencoder (VAE)** is a type of neural network used to learn latent representations of data, useful for NLP text generation and modeling. The VAE combines autoencoder techniques with probabilistic generative models, allowing new data samples similar to training ones to be generated. Their structure consists of an encoder that maps the input data into a probabilistic latent space and a decoder that reconstructs the original data from the points in the latent space.

The first steps are tokenization, creating token sequences, and adding padding to make them the same length.

```
from spacy.lang.it import Italian
import spacy
from nltk.lm.preprocessing import padded_everygram_pipeline
from nltk.tokenize.treebank import TreebankWordDetokenizer

import string

nlp_it = spacy.load("it_core_news_sm")
punctuations = string.punctuation
stop_words_it = spacy.lang.it.stop_words.STOP_WORDS
parser_it = Italian()
```

```
# Tokenizer function
def spacy_tokenizer_it(sentence):
    mytokens = parser_it(sentence)
    mytokens = [ word.text for word in mytokens ]
    # remove stop words
    mytokens = [ word for word in mytokens if word not in stop_words_it and word not in punctuations ]
    # return preprocessed list of tokens
    return mytokens
```

```
puglia = df[df['region']=='puglia']['text']
```

```
puglia
```

```
34        una grandissima artista barese ci lascia. add...
52        raffaele, mi sembra che sto'parlando con mio ...
59                                   bbeddhi comu lu sule
122         versione barese. la nonn gastema ! scritto da
325       la reazione di mio padre, da incorniciare, co...
                                ...
34208    A maje a diri ca a so' a bona persona, ma a ma...
34209    Mare e sole d'estate s'arriprende, a se'mpiede...
```

```
34210    A maje a diri ca a so' a persona onesta, ma a ...
34211    A se' a dispiaccie pe' chidd'ha pecato, a se' ...
34212    Cosa faje a sera quand'è freddo e nu viento 'n...
Name: text, Length: 1429, dtype: object
```

```
def get_sequence_of_tokens(corpus):
    word_index = {}
    index = 1
    input_sequences = []

    for line in corpus:
        token_list = spacy_tokenizer_it(line)
        token_indices = []
        for token in token_list:
            if token not in word_index:
                word_index[token] = index
                index += 1
            token_indices.append(word_index[token])

        for i in range(1, len(token_indices)):
            n_gram_sequence = token_indices[:i+1]
            input_sequences.append(n_gram_sequence)

    total_words = len(word_index) + 1
    return input_sequences, total_words,word_index

inp_sequences, total_words,word_index = get_sequence_of_tokens(puglia)
print("Sequence: ",inp_sequences[:10])
print("Total words: ",total_words)
print("Word index: ",word_index)
```

```
def generate_padded_sequences(input_sequences):
    max_sequence_len = max([len(x) for x in input_sequences])
    input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

    predictors, label = input_sequences[:,:-1],input_sequences[:,-1]
    label = ku.to_categorical(label, num_classes=total_words)
    return predictors, label, max_sequence_len

predictors, label, max_sequence_len = generate_padded_sequences(inp_sequences)
print("Predictors shape:", predictors.shape)
print("Label shape:", label.shape)
print("Max sequence length:", max_sequence_len)
```

Now we define the 3 parts of the VAE model:

- Encoder: first part of the model that serves to create the internal representation of each sequence that arrives. Each input will be transformed into two vectors representing it: mean vector and variance vector.
- Sampling: creation of internal input representation
- Decoder: for generating new text,dependent on the encoder's ouput.

```
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense, Lambda, RepeatVector, TimeDistributed
from tensorflow.keras.models import Model
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow.keras import backend as K
import numpy as np
```

```
input_dim = total_words  #vocabulary size
embedding_dim = 128      #embedding size
latent_dim = 64          #latent vector size
max_sequence_len = predictors.shape[1]  #maximum length of the sequences
```

*Definition of the encoder:*

```
def encoder(max_sequence_len,input_dim, embedding_dim,latent_dim):
  inputs = Input(shape=(max_sequence_len,))
  # Embedding Layer
  x = Embedding(input_dim, embedding_dim, input_length=max_sequence_len)(inputs)
  # LSTM Layer
  x = LSTM(128, return_sequences=False)(x)
  # Parameters of the latent distribution
  z_mean = Dense(latent_dim)(x)
  z_log_var = Dense(latent_dim)(x)

  return z_mean, z_log_var,inputs

z_mean, z_log_var,inputs = encoder(max_sequence_len,input_dim, embedding_dim,latent_dim)
```

*Definition of the sampling:*

```
def sampling(args):
    z_mean, z_log_var = args
    batch = K.shape(z_mean)[0]
    dim = K.int_shape(z_mean)[1]
    epsilon = K.random_normal(shape=(batch, dim))
    return z_mean + K.exp(0.5 * z_log_var) * epsilon


latent_vector = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_var])
```

*Definition of the decoder:*

```
def decoder(latent_vector,max_sequence_len):
  decoder_h = Dense(128, activation='relu')
  h_decoded = decoder_h(latent_vector)
  x_decoded_mean = RepeatVector(max_sequence_len)(h_decoded)
  x_decoded_mean = LSTM(128, return_sequences=True)(x_decoded_mean)
  x_decoded_mean = TimeDistributed(Dense(input_dim, activation='softmax'))(x_decoded_mean)

  return x_decoded_mean

x_decoded_mean =  decoder(latent_vector,max_sequence_len)
```

Definitive creation of the VAE model:

```
vae = Model(inputs, x_decoded_mean)


# Loss function
kl_weight = 0.1

reconstruction_loss = K.sum(K.sparse_categorical_crossentropy(inputs, x_decoded_mean), axis=-1)
kl_loss = 1 + z_log_var - K.square(z_mean) - K.exp(z_log_var)
kl_loss = K.sum(kl_loss, axis=-1)
kl_loss *= -0.5
vae_loss = K.mean(reconstruction_loss + kl_weight * kl_loss)
vae.add_loss(vae_loss)
vae.compile(optimizer='adam')


vae.fit(predictors, epochs=150 , batch_size=16, validation_split=0.1)
```

```
decoder_input = Input(shape=(latent_dim,))
decoder_h = Dense(128, activation='relu')
h_decoded = decoder_h(decoder_input)
x_decoded_mean = RepeatVector(max_sequence_len)(h_decoded)
x_decoded_mean = LSTM(128, return_sequences=True)(x_decoded_mean)
x_decoded_mean = TimeDistributed(Dense(input_dim, activation='softmax'))(x_decoded_mean)

decoder = Model(decoder_input, x_decoded_mean)

def generate_text(decoder, latent_dim, word_index, max_sequence_len, num_samples=1):
    sampled_latent_vectors = np.random.normal(size=(num_samples, latent_dim))

    decoded_sequences = decoder.predict(sampled_latent_vectors)

    index_word = {v: k for k, v in word_index.items()}

    generated_texts = []
    for seq in decoded_sequences:
        generated_text = ' '.join([index_word.get(index, '') for index in np.argmax(seq, axis=1)])
        generated_texts.append(generated_text)

    return generated_texts

generated_texts = generate_text(decoder, latent_dim, word_index, max_sequence_len, num_samples=5)
for i, text in enumerate(generated_texts):
    print(f"Generated text {i+1}: {text}")
```

```
1/1 [==============================] - 1s 914ms/step
Generated text 1: qual qual qual qual qual qual qual qual coscienza passa passa passa passa passa passa passa passa passa passa pas
Generated text 2: qual qual qual qual qual qual qual qual qual qual qual qual qual qual qual qual qual qual qual qual
Generated text 3: baci baci baci baci baci baci baci baci baci baci pur pur pur pur pur pur pur pur pur pur
Generated text 4: coscienz coscienz coscienz coscienz coscienz coscienz coscienz coscienz coscienz coscienz coscienz coscienz cosci
Generated text 5: raffaele raffaele raffaele raffaele raffaele raffaele raffaele raffaele raffaele raffaele raffaele raffaele raffa
```

As we can see, not even the VAE model works well for text generation. This is because of the limited data set. As a result, we try to use pre-addressed models.

## GEMINI-PRO

The first pre-trained model that was used is Gemini-pro via the API offered by Gemini

```
!pip install -q -U google-generativeai
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 164.2/164.2 kB 4.1 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 718.3/718.3 kB 8.8 MB/s eta 0:00:00
```

```
# Import the Python SDK
import google.generativeai as genai
# Used to securely store your API key
from google.colab import userdata

GOOGLE_API_KEY=userdata.get('GOOGLE_API_KEY')
genai.configure(api_key=GOOGLE_API_KEY)
```

```
model = genai.GenerativeModel('gemini-pro')
```

```
frasi = df[df['region']=='puglia']['text'][:10].values
regione = puglia
frasi
```

```
'[\' una grandissima artista barese ci lascia. addio a mariolina de fano. eccola qui che interpreta la vecchie e la mort via \'\n
" raffaele, mi sembra che sto\'parlando con mio figlio. quindi basta dire munnu e\' munnu sara\'. mi da fastidio, di chi non vuole
collaborare con l\'italia. "\n \'bbeddhi comu lu sule \' \' versione barese. la nonn gastema ! scritto da \'\n " la reazione di mi
o padre, da incorniciare, come al solito: ma tu vid nu picc, mo t\'aviva nca! pur sop a la coscienz\' t\'avevna tne l sant midc!
(trad.: guarda un po\' che adesso dovevi soffocarti! pure sulla coscienza ti dovevano tenere ermal e fabrizio!) "\n \'na brutta fa
c\'\n \' accendo la tv. papa guarda la tv. mi guarda: qual d l sant midc aspttam staser? ha gia capito tutto. \'\n \'none e fore t
```

```
prompt = "Scrivimi 10 frasi lunghe in dialetto della puglia dammele in csv"
```

```
response = model.generate_content(prompt)
print(response.text)
```

```
| Originale dialettale | Traduzione |
|---|---|
| "L'ave capite ca stè scurde 'ngule, uè?" | Hai capito che sta facendo buio qui, eh? |
```

```
| "Quidde uè, u bbène 'na fusse e u ddòmene ammure i' bbramme" | Quello lì, ti dà una botta e il giorno dopo ti tira le orecchie |
| "S'avìje fattende a u mare, purtatine 'ngule u paste de mandule" | Se andate al mare, portateci anche i pasticcini di mandorle |
| "U tiembbe dùre u bbène e ddùre u male, sta' sempre luatezze" | Il tempo dura sia il bene che il male, stai sempre attento |
| "Nun mbi ne scorde ca si' figliu meje, e 'u sange meje scorre 'nd'u core tue" | Non dimenticare mai che sei figlio mio, e che il
| "Acceppette uanne 'u diaule te chiame, e t'embie a ddumandà 'na ssande rrube" | Accetta quando il diavolo ti chiama, e ti manda a
| "U ciuane, quanne 'u uede u lupu, s'amminacce a u quaglie e 'u scìppe" | Il cucciolo, quando vede il lupo, minaccia la quaglia e
| "A gghie jeu, ca mbi si' runate te 'ssole, ca te vèje sèche a lu sole" | Io che sono la tua rovina del sole, che ti vedo seccare
| "Te vuèje bbene, ma cchiù 'nta u core meje ca 'nta a u ccape tue" | Ti voglio bene, ma più nel mio cuore che nella tua testa |
| "U tiembbe ca trase 'nd'a nu core, lassene 'na cumme cchiù 'nta a nu juore" | Il tempo che entra in un cuore, lascia una ferita p
```

It doesn’t work bad.

## GPT-2

To get more precision, let’s also try GPT-2. On it is applied a fine-tuning phase for each dialect.

```
!pip install transformers[torch]
```

Mostra output nascosti

```
!pip install accelerate -U
```

Mostra output nascosti

```
import pandas as pd
from transformers import GPT2LMHeadModel, GPT2Tokenizer, Trainer, TrainingArguments, TextDataset, DataCollatorForLanguageModeling
from sklearn.model_selection import train_test_split
import os

regions = df['region'].unique()

for region in regions:
    region_df = df[df['region'] == region]
    texts = region_df['text'].tolist()

    train_texts, val_texts = train_test_split(texts, test_size=0.1, random_state=42)

    train_file = f'train_{region}.txt'
    val_file = f'val_{region}.txt'

    with open(train_file, 'w') as f:
        f.write('\n'.join(train_texts))

    with open(val_file, 'w') as f:
        f.write('\n'.join(val_texts))

    def load_dataset(train_path, val_path, tokenizer):
        train_dataset = TextDataset(
            file_path=train_path,
            tokenizer=tokenizer,
            block_size=128
        )
        val_dataset = TextDataset(
            file_path=val_path,
            tokenizer=tokenizer,
            block_size=128
        )
        return train_dataset, val_dataset

    model_name = 'gpt2'
    tokenizer = GPT2Tokenizer.from_pretrained(model_name)
    model = GPT2LMHeadModel.from_pretrained(model_name)

    train_dataset, val_dataset = load_dataset(train_file, val_file, tokenizer)

    data_collator = DataCollatorForLanguageModeling(
        tokenizer=tokenizer,
        mlm=False,
    )

    training_args = TrainingArguments(
        output_dir=f'./results_{region}',
        overwrite_output_dir=True,
        num_train_epochs=3,
        per_device_train_batch_size=4,
        save_steps=10_000,
        save_total_limit=2,
        prediction_loss_only=True,
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        data_collator=data_collator,
        train_dataset=train_dataset,
        eval_dataset=val_dataset,
    )

    trainer.train()

    model.save_pretrained(f'./fine_tuned_model_{region}')
    tokenizer.save_pretrained(f'./fine_tuned_model_{region}')
```

Mostra output nascosti

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer

def generate_sentence(region, input_text, max_length=100, num_return_sequences=1, top_k=50, top_p=0.95, temperature=0.7):
    model_path = f'./fine_tuned_model_{region}'
    tokenizer = GPT2Tokenizer.from_pretrained(model_path)
    model = GPT2LMHeadModel.from_pretrained(model_path)

    model.eval()

    input_ids = tokenizer.encode(input_text, return_tensors='pt')

    output = model.generate(
        input_ids,
        max_length=max_length,
        num_return_sequences=1,
        top_k=top_k,
        top_p=top_p,
        temperature=temperature
    )

    generated_text = tokenizer.decode(output[0], skip_special_tokens=True)

    return generated_text

region = "valle d'aosta"
input_text = "ciao"
generated_sentence = generate_sentence(region, input_text)
print(generated_sentence)
```

```
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass your input
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
ciao, a former student of the Chinese Communist Party, said that the party's policy of "reform" was "a very serious mistake."

"The party's policy of reform is a very serious mistake," he said. "It is a very serious mistake. It is a very serious mistake. It

The party's policy of "reform" is a very serious mistake. It is a very serious mistake. It is a very serious mistake
```

After a few attempts, GPT-2 doesn't work so badly, but it always remains an English-based model and sometimes it doesn't meet the task and responds in English. So, let's try to implement a model that has already been fine-tuned for the Italian language

## LLaMAntino-3-ANITA-8B-Inst-DPO-ITA

The selected model is **LLaMAntino-3-ANITA-8B-Inst-DPO-ITA** which is a large language model developed for advanced natural language processing (NLP) applications in Italian, such as text generation, machine translation, completion of sentences and answers to questions. Thanks to its 8 billion parameters, it can handle complex tasks and provide more accurate and contextually relevant answers.

```
!pip install pyarrow<15.0.0a0,>=14.0.1
!pip install requests==2.31.0
!pip install pyarrow >=2
!pip install -U transformers trl peft accelerate bitsandbytes
```

Mostra output nascosti

```
df_initial= pd.read_csv("NLP_DatasetPrima.csv")
df_prov = pd.read_csv("NLP_Dataset_OS.csv")
```

```
#the model upload

import torch
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
)

base_model = "swap-uniba/LLaMAntino-3-ANITA-8B-Inst-DPO-ITA"
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
    bnb_4bit_use_double_quant=False,
)

model = AutoModelForCausalLM.from_pretrained(
    base_model,
    quantization_config=bnb_config,
    device_map="auto",
)

tokenizer = AutoTokenizer.from_pretrained(base_model)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarn
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (http
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public m
  warnings.warn(
```

| | | |
|---|---|---|
| config.json: 100% | 654/654 | [00:00<00:00, 16.0kB/s] |
| model.safetensors.index.json: 100% | 23.9k/23.9k | [00:00<00:00, 527kB/s] |
| Downloading shards: 100% | 4/4 | [02:27<00:00, 31.12s/it] |
| model-00001-of-00004.safetensors: 100% | 4.98G/4.98G | [00:52<00:00, 185MB/s] |
| model-00002-of-00004.safetensors: 100% | 5.00G/5.00G | [00:39<00:00, 74.4MB/s] |
| model-00003-of-00004.safetensors: 100% | 4.92G/4.92G | [00:45<00:00, 170MB/s] |
| model-00004-of-00004.safetensors: 100% | 1.17G/1.17G | [00:09<00:00, 37.8MB/s] |
| Loading checkpoint shards: 100% | 4/4 | [01:06<00:00, 14.35s/it] |
| generation_config.json: 100% | 182/182 | [00:00<00:00, 11.9kB/s] |

```
sys = "Sei un assistente digitale AI per la lingua dialettale italiana di nome LLaMAntino-3 ANITA." \
    "(Advanced Natural-based interaction for the ITAlian language)." \
    " Rispondi imitando il linguaggio con cui ti vengono passate le frasi."

import transformers
pipe = transformers.pipeline(
    model=model,
    tokenizer=tokenizer,
    return_full_text=False, # langchain expects the full text
    task='text-generation',
    max_new_tokens=512, # max number of tokens to generate in the output
    temperature=0.6,  #temperature for more or less creative answers
    do_sample=True,
    top_p=0.9,
)


text = df_initial[df_initial['region']=='puglia']['text'].tolist()
prompt = f"Le frasi delimitate da \' sono frasi del dialetto di Puglia: \'{text[0]}\',\'{text[1]}\',\'{text[2]}\',\'{text[3]}\',\'{text


messages = [
    {"role": "system", "content": sys},
    {"role": "user", "content": prompt}
]

sequences = pipe(messages)
for seq in sequences:
    print(f"{seq['generated_text']}")
```

```
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
Cu' 'na gran'de pessime notti invernali
Fa difìcile usci' a fa' someje
Munnu ca s'addimmora, s'addimmora
Nn'è cchiù roba a fà, si s'addorme
E ccà ven' a fà dispiacere a mamma
Nn'è 'nu omme ca no' pecca, pecca pure 'o santo
T'aspetta 'a st'anne e t'aspetta 'n'altra
Cchiù mali ca bene, cchiù mali ca bene
Facc' a mme a penza, a mme a penza
Chiddh' ca s'innamora, s'innamora a l'immagine';
```

Among the generation models, this is the most suitable. Then, in the next step, we will use LLaMAntino-3-ANITA-8B-Inst-DPO-ITA for generating new sentences.

## OVER-SAMPLING

```
df_prov = pd.read_csv("NLP_Dataset_OS.csv")
print(df_prov.shape)

add = {"text": [], "region": []}

regions = ['veneto','lombardia','sicilia','toscana','sardegna','emilia romagna','calabria','puglia','piemonte','liguria','friuli-venezi

for region in regions:
    texts = df_initial[df_initial['region'] == region]['text'].tolist()

    random_choose = []
    for _ in range(7):
      random_choose.append(randint(0, len(texts)-1))

    prompt = f"Le frasi delimitate da \' sono frasi del dialetto della regione {region}: \'{texts[random_choose[0]]}\',\'{texts[random_

    print(prompt)
    print("\n")

    messages = [
        {"role": "system", "content": sys},
        {"role": "user", "content": prompt}
    ]

    sequences = pipe(messages)

    for seq in sequences:
        generated_text = seq['generated_text']
        print(f"{seq['generated_text']}")

        # Split phrases into separate rows (if necessary)
        if "\n" in generated_text:
            phrases = generated_text.split("\n")
            add["text"].extend(phrases)
            add["region"].extend([region] * len(phrases))
        else:
            add["text"].append(generated_text)
            add["region"].append(region)
```

Mostra output nascosti

```
df_new = pd.DataFrame(add)
df_combined = pd.concat([df_prov, df_new], ignore_index=True)
df_combined['region'].value_counts()
```

```
region
abruzzo                 80
sicilia                 65
calabria                62
marche                  59
lombardia               57
piemonte                54
umbria                  54
basilicata              53
friuli-venezia giulia   52
trentino-alto adige     50
sardegna                46
veneto                  44
emilia romagna          43
molise                  42
toscana                 41
```

```
puglia                  40
valle d'aosta           31
liguria                 29
Name: count, dtype: int64
```

```
df_combined.to_csv('NLP_Dataset_OS.csv',index=False)
```

```
df_combined.shape
```

```
(902, 2)
```

```
df_OS = pd.read_csv("Dataset_Quarto.csv")
df_combined = pd.concat([df_OS,df_combined], ignore_index=True)
df_combined.to_csv('Dataset_Quarto.csv',index=False)
df_combined['region'].value_counts()
```

```
region
lazio                    5614
campania                 3046
veneto                   1979
sicilia                  1848
lombardia                1802
toscana                  1649
sardegna                 1535
calabria                 1508
puglia                   1429
piemonte                 1413
friuli-venezia giulia    1387
emilia romagna           1379
marche                   1321
liguria                  1319
basilicata               1304
abruzzo                  1300
umbria                   1246
trentino-alto adige      1226
molise                   1226
valle d'aosta             948
Name: count, dtype: int64
```

```
df_combined.shape
```

```
(33577, 3)
```

After several iterations we managed to make the dataset bigger with the addition of 19,754 examples going also to increase the examples of the Minonitary classes.

## TASK: Classification of sentences by region

The main task that you want to satisfy in this project is the classification of dialect phrases by region. To do this, an SVM classifier from SKlearn is implemented.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
from imblearn.pipeline import Pipeline as ImbPipeline
from sklearn.base import TransformerMixin
from statistics import mean, stdev
from sklearn import preprocessing
from sklearn.model_selection import StratifiedKFold
```

```
df = pd.read_csv("Dataset_Quarto.csv")
df = df.drop(['Unnamed: 0'],axis = 1)
df
```

| | text | region |
|---|---|---|
| 0 | il chiosco bar e dove si e co i lettini,appen... | lazio |
| 1 | so' monticiano, so', sangue de zio! so' nato ... | lazio |
| 2 | veneziani, gran signori; padovani, gran dotor... | veneto |
| 3 | poi se bu avanzanu zeppule passati de casa ca ... | calabria |
| 4 | come disse n'amica mia anni fa, alla seconda f... | lazio |
| ... | ... | ... |
| 34474 | Quand a l'é andà a scola, a l'ha pasàa la part... | valle d'aosta |
| 34475 | A l'é andà a caccia, e a l'é tornà a cà, e a l... | valle d'aosta |
| 34476 | I doi, i l'era andà a spasseggià, e a l'é stac... | valle d'aosta |
| 34477 | A l'é mòrt a sò nonno, a l'ha dàita un pò' de ... | valle d'aosta |
| 34478 | A l'é andà a fè la rixe, e a l'é tornà a cà, e... | valle d'aosta |

34479 rows × 2 columns

```
#decoding text from latin-1 to utf-8
df['text'] = df['text'].apply(lambda x: x.decode('latin-1') if isinstance(x, bytes) else x)
#replacing Nan values with empty strings
df['text'].fillna('', inplace=True)
```

```
X = df['text'].to_numpy()
y = df['region'].to_numpy()
```

```
textclassifier = ImbPipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('mnb', SVC())
])
```

```
from sklearn import metrics

n_splits = 20

fmacro = 0
fmicro = 0
facc = 0
frecall = 0
fprecision = 0
y_gt= []
y_pred = []

skf = StratifiedKFold(n_splits=n_splits, shuffle=True)

for train_index, test_index in skf.split(X, y):
  x_train_fold, x_test_fold = X[train_index], X[test_index]
  y_train_fold, y_test_fold = y[train_index], y[test_index]
  textclassifier.fit(x_train_fold, y_train_fold)
  pred = textclassifier.predict(x_test_fold)

  y_gt.extend(y_test_fold)
  y_pred.extend(pred)

  # Valutation metrics
  fmacro += metrics.f1_score(y_test_fold, pred, average='macro')
  fmicro += metrics.f1_score(y_test_fold, pred, average='micro')
  facc += metrics.accuracy_score(y_test_fold, pred)
  fprecision += metrics.precision_score(y_test_fold, pred, average='macro')
  frecall += metrics.recall_score(y_test_fold, pred, average='macro')

print("\n=========================================================================================================================
print("Accuracy:", facc/n_splits)
print("P={0}, R={1}, F1 Macro={2}, F1 Micro={2}".format(fprecision/n_splits, frecall/n_splits, fmacro/n_splits, fmicro/n_splits))
print("=========================================================================================================================
print(metrics.classification_report(y_gt, y_pred, digits=2))
```

```
==================================================================================================================================
Accuracy: 0.7005137770278733
P=0.7101501776285007, R=0.6471214558008442, F1 Macro=0.671757058855882, F1 Micro=0.671757058855882
==================================================================================================================================
                       precision    recall  f1-score   support
```

```
               abruzzo       0.64      0.51      0.57      1300
            basilicata       0.68      0.58      0.63      1304
              calabria       0.70      0.57      0.63      1508
              campania       0.78      0.85      0.81      3046
        emilia romagna       0.61      0.50      0.55      1379
 friuli-venezia giulia       0.72      0.66      0.69      1387
                 lazio       0.63      0.96      0.76      5614
               liguria       0.77      0.61      0.68      1319
             lombardia       0.70      0.60      0.64      1802
                marche       0.66      0.51      0.57      1321
                molise       0.66      0.57      0.61      1226
              piemonte       0.68      0.61      0.64      1413
                puglia       0.69      0.60      0.64      1429
              sardegna       0.91      0.85      0.88      1535
               sicilia       0.76      0.79      0.77      1848
               toscana       0.71      0.65      0.68      1649
   trentino-alto adige       0.71      0.60      0.65      1226
                umbria       0.65      0.51      0.57      1246
         valle d'aosta       0.77      0.65      0.71       948
                veneto       0.73      0.78      0.75      1979

              accuracy                           0.70     34479
             macro avg       0.71      0.65      0.67     34479
          weighted avg       0.70      0.70      0.69     34479
```

```
textclassifier.predict(["c'ama sci sciamanin"])
```

```
array(['puglia'], dtype=object)
```

We can see the results obtained. We can see that:

- Accuracy: The model achieved an overall accuracy of 70%, indicating that 70% of the phrases were correctly classified.
- Recall medium: The average recall is 65%, indicating that the model is moderately effective in capturing all dialect phrases for each region.
- F1 average score: The F1 average score is 67%, which represents a good balance between accuracy and recall.

In detail we have that:

- The dialect of Sardinia is classified more precisely.
- The dialect of the Emilia-Romagna region is classified with less precision.

## EXTRA TASK: Translation of dialect phrases

Another experiment that has been conducted within this project is to test Lamantino's performance in translating dialect phrases. To conduct this experiment was first loaded the model and defined its role within this task.

```
!pip install tqdm
from tqdm import tqdm
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (4.66.4)
```

```
sys = "Sei un assistente digitale AI per la lingua dialettale italiana di nome LLaMAntino-3 ANITA." \
    "(Advanced Natural-based interaction for the ITAlian language)." \
    "Traduci in italiano le espressioni dialettali che si trovano all'interno del testo fornito dall'utente,ma senza cambiare il signif

import transformers
pipe = transformers.pipeline(
    model=model,
    tokenizer=tokenizer,
    return_full_text=False, # langchain expects the full text
    task='text-generation',
    max_new_tokens=512, # max number of tokens to generate in the output
    temperature=0.5,  #temperature for more or less creative answers
    do_sample=True,
    top_p=0.9
)
```

```
import pandas as pd

df= pd.read_csv("Dataset_Quarto.csv")
add = {"trad": []}
df = df.drop(['Unnamed: 0'],axis = 1)
apulia = df[df['region']=='puglia']
```

In order to evaluate the quality of the translation, the task requires you to have a dataset containing manual translations, made by local people, of the sentences. So you can compare them with machine translations. For time reasons, 100 sentences belonging to the Apulian dialect have been manually and automatically translated.

```
apulia = apulia[:100]
apulia
```

| | text | region |
|---|---|---|
| 34 | una grandissima artista barese ci lascia. add... | puglia |
| 52 | raffaele, mi sembra che sto'parlando con mio ... | puglia |
| 59 | bbeddhi comu lu sule | puglia |
| 122 | versione barese. la nonn gastema ! scritto da | puglia |
| 325 | la reazione di mio padre, da incorniciare, co... | puglia |
| ... | ... | ... |
| 5490 | te mpauri de mie tie ahahhah sine sine la porto | puglia |
| 5530 | anche perche no je manc sicur ca riman idd | puglia |
| 5603 | lu sule c'e. lu mare c'e... lu jentu...no... | puglia |
| 5606 | aggiu' capito michele stai passando, con: cara... | puglia |
| 5659 | stefano reali tutt tu tutt tu e fasc sti ralle... | puglia |

100 rows × 2 columns

```
for sentence in tqdm(apulia.to_numpy()):
  prompt = f"""Testo: "{sentence[0]}".
  Il testo è scritto nel dialetto della regione italiana {sentence[1]},
  traducilo in lingua italiana senza cambiare nè la sua semantica nè la sua sintassi.
  Termina la traduzione con un "\n".
  Non devono essere date in output altre informazioni oltre la traduzione.
  Non aggiungere parentesi o altri commenti. Non aggiungere parole inglesi o italiane che non siano già presenti nel testo.
  Lascia invariati i termini che sono già all'interno del testo in lingua italiana o inglese.
  Rispondi solo con la traduzione letterale.
  Non saltare nessuna parte del testo. Neanche quelle che originariamente sono tra perentesi nel testo.
  """

  messages = [
      {"role": "system", "content": sys},
      {"role": "user", "content": prompt}
  ]

  sequences = pipe(messages)

  for seq in sequences:
      generated_text = seq['generated_text']
      print(f"{seq['generated_text']}")

      # Split phrases into separate rows (if necessary)
      if "\n" in generated_text:
          phrases = generated_text.split("\n")
          add["trad"].extend(phrases)
      else:
          add["trad"].append(generated_text)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
  1%|          | 1/100 [00:05<08:50,  5.36s/it]Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
Una grande artista barese ci lascia. Addio a Mariolina de Fano. Eccola qui che interpreta la vecchia e la morte. "
  2%|▏         | 2/100 [00:11<09:28,  5.80s/it]Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
Raffaele, mi sembra di dire cose che dicono mio figlio. Quindi basta dire: il mondo è il mondo sarà'. Mi dà fastidio, di chi non
```