DeepSphere.AI
Enterprise AI and IIoT for Analytics

MCQ GENERATION

Using Machine Learning

TABLE OF CONTENTS

DISCLAIMER

All software and hardware used or referenced in this guide belong to their respective vendor. We developed this guide based on our development infrastructure and this guide may or may not work on others systems and technical infrastructure. We are not liable for any direct or indirect problems caused by users using this guide.

EXECUTIVE SUMMARY

The purpose of this document is to provide adequate information to users to implement a Supervised Machine Learning model. In order to achieve this, we are using one of the most common problem that occurs at Educational Institutions. Traditionally, a Teacher picks a few questions from the Chapter and these questions are repeated every year. Each Chapter will require its own way of Learning and Understanding. In order to generate questions in all Dimensions Possible for every Chapter, we use Machine Learning Techniques, so that Machine can Learn, Understand and Frame as many questions as possible instead of a Teacher taking questions by themselves.

Confidential and Proprietary | © DeepSphere.AI, Inc. | Step by Step Lab Guide | v1
DeepSphere.AI | DsSchoolOfAI.Com | Info@DeepSphere.AI

BUSINESS PROBLEM

Confidential and Proprietary | © DeepSphere.AI, Inc. | Step by Step Lab Guide | v1
DeepSphere.AI | DsSchoolOfAI.Com | Info@DeepSphere.AI

DeepSphere.AI
Enterprise AI and IIoT for Analytics

PROBLEM STATEMENT

Given the Paragraph, Frame Multiple Choice Questions and Generate
Distractors (Options) similar to the Answer.

BUSINESS CHALLENGES

- Takes more time
- Human intensive
- The questions generated by humans are limited to predefined context

BUSINESS CONTEXT

In an Educational Institution, a teacher picks few Questions from a Chapter. These Questions are repeated every Semester, for example, a Maths Teacher creates few types of Questions for a chapter, it is repeated for the students passing that semester for many years. The learning speed of each student varies, there can't be common questions across all the students. So the questions has to be dynamically driven. In order to achieve this, we are using Machine Learning Technique, so that the Machines can learn, understand and create Questions in all Dimensions Possible by itself instead of a Teacher picking questions by themselves.

DATA MANAGEMENT

There are three types of data sets Training, Test and Dev that are used at various stage of Implementation. Training dataset is the largest of three of them, while test data functions as seal of approval and you don't need to use till the end of the development.

DeepSphere.AI
Enterprise AI and IIoT for Analytics

TRAINING DATASET

The training data set is the actual dataset used to train the model for performing various Machine Learning Operations (Regression, Classification, Clustering etc.). This is the actual data with which the models learn with various API and algorithm to train the machine to work automatically.

	A	B	C	D	E	F	G	H	I	J
1	title	context	question	answer	answer_st	answer_e	answer_st	answer_end_word_idx		
2	University	Architectu	To whom	Saint Bern	515	541	102	104		
3	University	Architectu	What is in	a copper s	188	213	37	41		
4	University	Architectu	The Basilic	the Main E	279	296	57	59		
5	University	Architectu	What is th	a Marian p	381	420	76	82		
6	University	Architectu	What sits	a golden s	92	126	17	23		
7	University	As at mos	When did	Septembe	248	262	45	46		
8	University	As at mos	How ofter	twice	441	446	78	78		
9	University	As at mos	What is th	The Obser	598	610	216	217		
10	University	As at mos	How man	three	126	131	222	222		
11	University	As at mos	In what ye	1987	908	912	158	158		
12	University	The unive	Where is t	Rome	119	123	22	22		
13	University	The unive	What is th	Moreau S	145	160	121	122		
14	University	The unive	What is th	Old Colleg	234	245	46	47		
15	University	The unive	What indi	Retired pr	356	384	68	71		
16	University	The unive	Which pri	Buechner	675	703	125	128		
17	University	The Colleg	How man	eight	487	492	79	79		
18	University	The Colleg	In what ye	1920	46	50	7	7		
19	University	The Colleg	Before the	the Colleg	126	148				

```
the -0.038194 -0.24487 0.72812 -0.39961 0.083172 0.043953 -0.39141 0.3344 -0.57545 0.087459
, -0.10767 0.11053 0.59812 -0.54361 0.67396 0.10663 0.038867 0.35481 0.06351 -0.094189 0.15786
. -0.33979 0.20941 0.46348 -0.64792 -0.38377 0.038034 0.17127 0.15978 0.46619 -0.019169
of -0.1529 -0.24279 0.89837 0.16996 0.53516 0.48784 -0.58826 -0.17982 -1.3581 0.42541 0.15377
to -0.1897 0.050024 0.19084 -0.049184 -0.089737 0.21006 -0.54952 0.098377 -0.20135 0.34241
and -0.071953 0.23127 0.023731 -0.50638 0.33923 0.1959 -0.32943 0.18364 -0.18057 0.28963
in 0.085703 -0.22201 0.16569 0.13373 0.38239 0.35401 0.01287 0.22461 -0.43817 0.50164 -0.35874
a -0.27086 0.044006 -0.02026 -0.17395 0.6444 0.71213 0.3551 0.47138 -0.29637 0.54427 -0.72294
" -0.30457 -0.23645 0.17576 -0.72854 -0.28343 -0.2564 0.26587 0.025309 -0.074775 -0.3766
's 0.58854 -0.2025 0.73479 -0.68338 -0.19675 -0.1802 -0.39177 0.34172 -0.60561 0.63816
for -0.14401 0.32554 0.14257 -0.099227 0.72536 0.19321 -0.24188 0.20223 -0.89599 0.15215
- -1.2557 0.61036 0.56793 -0.96596 -0.45249 -0.071696 0.57122 -0.31292 -0.43814 0.90622
that -0.093337 0.19043 0.68457 -0.41548 -0.22777 -0.11803 -0.095434 0.19613 0.17785 -0.020244
on -0.21863 -0.42664 0.5196 0.0043103 0.58045 -0.10873 -0.37726 0.4566 -0.60627 -0.075773
is -0.54264 0.41476 1.0322 -0.40244 0.46691 0.21816 -0.074864 0.47332 0.080996 -0.22079
was 0.13717 -0.54287 0.19419 -0.29953 0.17545 0.084672 0.67752 0.098295 -0.035611 0.21334
said -0.13128 -0.452 0.043399 -0.99798 -0.21053 -0.95868 -0.24609 0.48413 0.18178 0.475
with -0.43608 0.39104 0.51657 -0.13861 0.2029 0.50723 -0.012544 0.22948 -0.6316 0.21199
he 0.1225 -0.058833 0.23658 -0.28877 -0.028181 0.31524 0.070229 0.16447 -0.027623 0.25214
```

TEST DATASET

Test data set helps you to validate that the training has happened efficiently in terms of either accuracy, or precision so on. Actually, such data is used for testing the model whether it is responding or working appropriately or not.

MCQ_AI_DATA.txt - Notepad

File Edit Format View Help

Artificial Intelligence refers to the intelligence of machines. This is in contrast to the natural intelligence of humans and animals. With Artificial Intelligence, machines perform functions such as learning, planning, reasoning and problem-solving. Most noteworthy, Artificial Intelligence is the simulation of human intelligence by machines. It is probably the fastest-growing development in the World of technology and innovation. Furthermore, many experts believe AI could solve major challenges and crisis situations. Artificial Intelligence in business would significantly save time and effort. There is an application of robotic automation to human business tasks. Furthermore, Machine learning algorithms help in better serving customers. Chatbots provide immediate response and service to customers.

MACHINE LEARNING LIBRARIES USED

1	spaCy
2	Pandas
3	Gensim
4	Random
5	Pickle

MODEL USED:

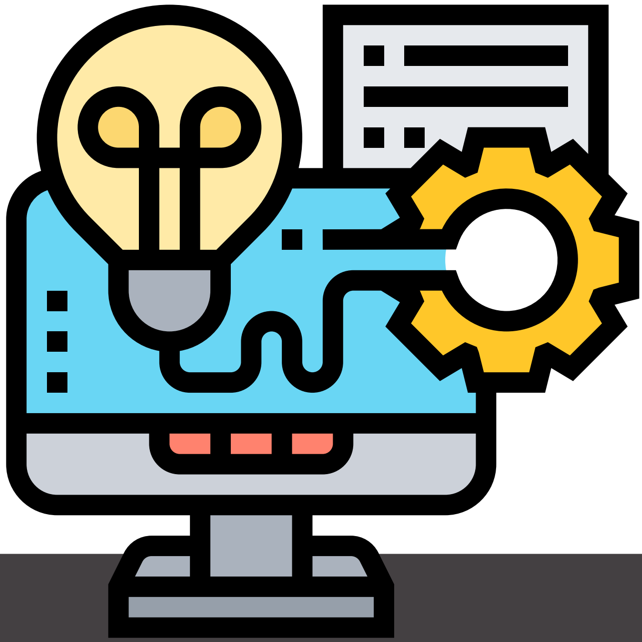
Gaussian Naive Bayes

MODEL BUILDING BLOCKS

There are several technical and functional components involved in implementing this model. Here are the key building blocks to implement the model.

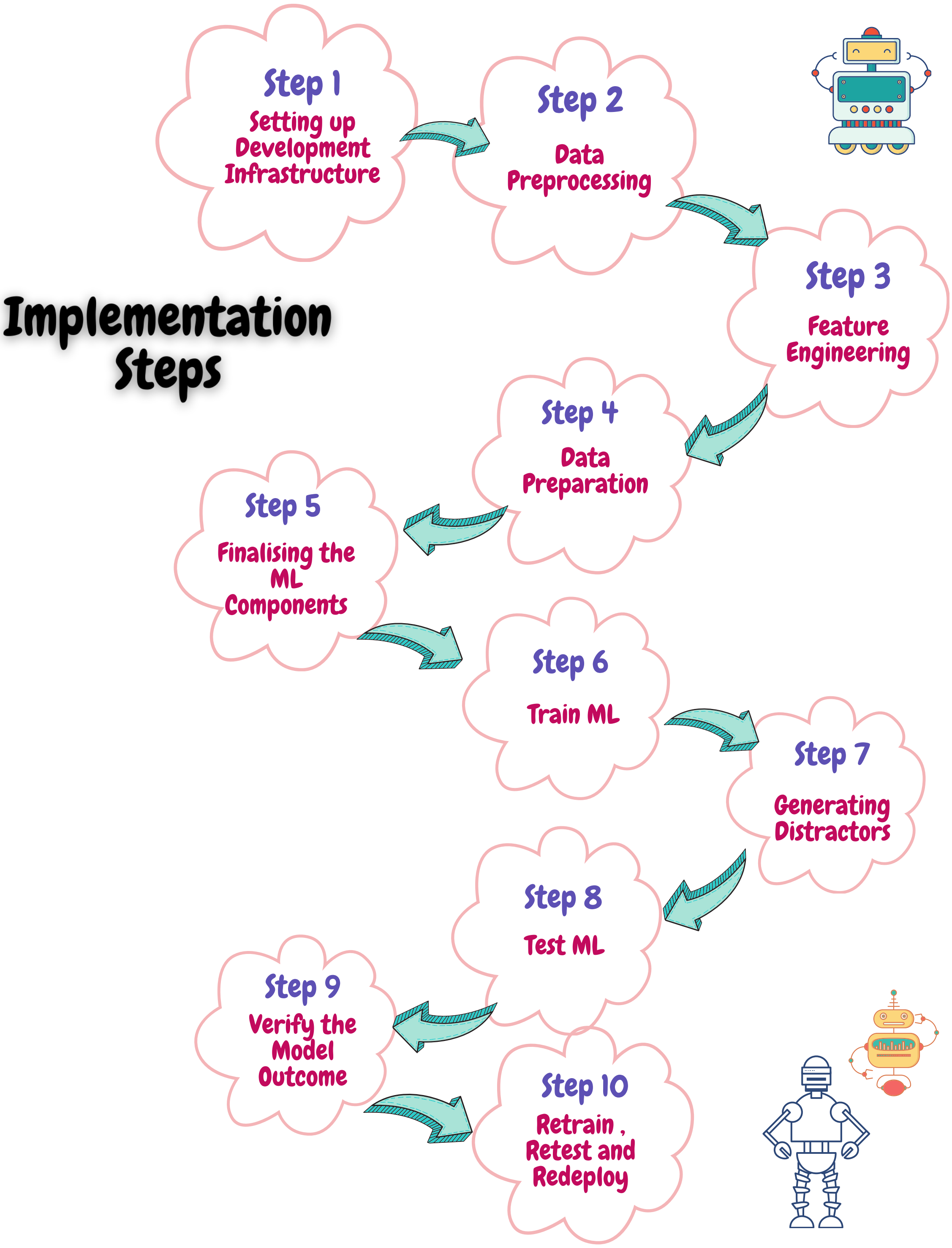
MACHINE LEARNING BUILDING BLOCKS

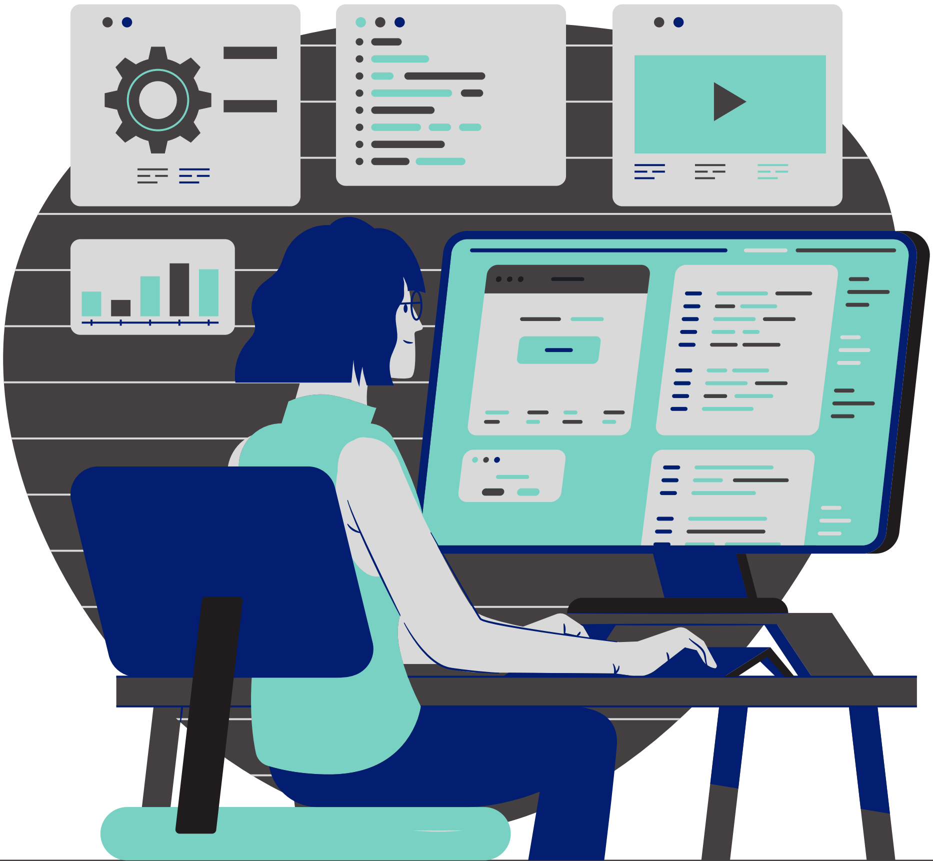
1. GOOGLE CLOUD
2. PYTHON
3. JUPYTER NOTEBOOK
4. PANDAS
5. PYSPARK - DATA ENGINEERING
6. GENSIM
7. PICKLE
8. RANDOM
9. SPACY



MACHINE LEARNING IMPLEMENTATION STEPS

Here are the key steps that are involved to implement a deep learning model. You can customize these steps as needed and we have developed these steps for learning-purpose only.


Implementation Steps
Step 1
Setting up Development Infrastructure
Step 2
Data Preprocessing
Step 3
Feature Engineering
Step 4
Data Preparation
Step 5
Finalising the ML Components
Step 6
Train ML
Step 7
Generating Distractors
Step 8
Test ML
Step 9
Verify the Model Outcome
Step 10
Retrain , Retest and Redeploy



ML MODEL IMPLEMENTATION STEPS

1

STEP 1-SETTING UP DEVELOPMENT INFRASTRUCTURE:

For our Model Implementation we need theFollowing Libraries:

Pandas: Pandas is a library used for data manipulation and analysis. For Our Implementation we are using it for Importing the Data file & Creating Dataframes (Stores the Data).

Spacy: It can be used to build information extraction or natural language understanding systems, or to pre-process text for deep learning. It can be used to build information extraction or natural language understanding systems, or to pre-process text for deep learning.

Gensim: Gensim is a Python library for topic modelling, document indexing and similarity retrieval with large corpora.

Random: Python offers random module that can generate random numbers. These are pseudo-random number
as the sequence of number generated depends on the seed. If the seeding value is same, the sequence will be the same.

DeepSphere.AI
Enterprise AI and IIoT for Analytics

mcq.py

```
/**********************************************************************

FILE NAME     : mcq_question_generation.py
Purpose       : To generate a paragraph of text automatically
Author        : DeepSphere.AI, Inc.
Date and Time : 12/07/2020 10:00 hrs
Version       : 1.0

/**********************************************************************

#############################################
# Step 1 - Import the Required Libraries    #
#############################################

import pandas as pd
from IPython.display import Markdown, display, clear_output
import spacy
from spacy import displacy
import _pickle as cPickle
from pathlib import Path
import gensim
from gensim.test.utils import datapath, get_tmpfile
from gensim.models import KeyedVectors
import random

#############################################
#    Step 2-Import Clustering Data          #
#############################################
vAR_Pickle_Data1 =C:\AI\AUTOMATIC QUESTION GENERATION\ML\TRAINING DATA\pickles\nb-predictor.pkl
vAR_Pickle_Data2 =C:\AI\AUTOMATIC QUESTION GENERATION\ML\TRAINING DATA\pickles\wordsDf.pkl
vAR_Training_Data1 = C:\AI\AUTOMATIC QUESTION GENERATION\ML\TRAINING DATA\embeddings\glove.6B.300d.txt
vAR_Training_Data2 =C:\AI\AUTOMATIC QUESTION GENERATION\ML\TRAINING DATA\embeddings\word2vec-glove.6B.300d.txt
vAR_Test_Data = C:\AI\AUTOMATIC QUESTION GENERATION\ML\TEST DATA\MCQ_AI_DATA.txt
```

2

STEP 2 - DATA PREPROCESSING

Next immediate step after importing all libraries is Data preprocessing i.e. Pickling. "Pickling" is the process whereby a Python object hierarchy is converted into a byte stream.

```python
############################################
#              Step 3-Pickling             #
############################################

def dumpPickle(fileName, content):
    pickleFile = open(fileName, 'wb')
    cPickle.dump(content, pickleFile, -1)
    pickleFile.close()

def loadPickle(fileName):
    file = open(fileName, 'rb')
    content = cPickle.load(file)
    file.close()
    return content

def pickleExists(fileName):
    file = Path(fileName)

    if file.is_file():
        return True

    return False

```

3

STEP 3 - FEATURE ENGINEERING

Step 3 of the Implementation is Feature Generation/ Feature Engineering. Machine learning works on a simple rule – if you put garbage in, you will only get garbage to come out. By garbage here, we mean noise in data. This becomes even more important when the numbers of features are very large. We need only those features (Input) that are function of the Labels (Outputs). Ex: To Predict whether the given fruit is an apple or orange Color/Texture of the Fruit becomes a feature to be Considered. If the Color/Texture is Red then it an Apple, If it's Orange its Orange.

```
#########################################################################
#              Step 4-Extract Words and Generate Features               #
#########################################################################

vAR_model = LinearRegression()
vAR_model.fit(vAR_Features_Train,vAR_Labels_Train)
import en_core_web_sm
nlp = spacy.load('en_core_web_sm')

################################
#Extract answers and the sentence they are in
################################
def extractAnswers(qas, doc):
    answers = []

    senStart = 0
    senId = 0

    for sentence in doc.sents:
        senLen = len(sentence.text)

        for answer in qas:
            answerStart = answer['answers'][0]['answer_start']

            if (answerStart >= senStart and answerStart < (senStart + senLen)):
                answers.append({'sentenceId': senId, 'text': answer['answers'][0]['text']})

        senStart += senLen
        senId += 1

    return answers
################################
# Cleaning answers from stopwords
################################
def tokenIsAnswer(token, sentenceId, answers):
    for i in range(len(answers)):
        if (answers[i]['sentenceId'] == sentenceId):
            if (answers[i]['text'] == token):
                return True
    return False
```

4

STEP 4 - DATA PREPERATION

Step 4 involved fixing named entities start points. In information extraction, a named entity is a real-world object, such as persons, locations, organizations, products, etc., that can be denoted with a proper name. It can be abstract or have a physical existence. Hence, These named entities can be used to select potential Questions and answers.

```python
################################################
#    Step 5-Fixing named entities start points #
################################################

################################
#Save named entities start points
################################

def getNEStartIndexs(doc):
    neStarts = {}
    for ne in doc.ents:
        neStarts[ne.start] = ne

    return neStarts

def getSentenceStartIndexes(doc):
    senStarts = []

    for sentence in doc.sents:
        senStarts.append(sentence[0].i)

    return senStarts

def getSentenceForWordPosition(wordPos, senStarts):
    for i in range(1, len(senStarts)):
        if (wordPos < senStarts[i]):
            return i - 1

```

```
def addWordsForParagrapgh(newWords, text):
    doc = nlp(text)

    neStarts = getNEStartIndexs(doc)
    senStarts = getSentenceStartIndexes(doc)

    #index of word in spacy doc text
    i = 0

    while (i < len(doc)):
        #If the token is a start of a Named Entity, add it and push to index to end of the NE
        if (i in neStarts):
            word = neStarts[i]
            #add word
            currentSentence = getSentenceForWordPosition(word.start, senStarts)
            wordLen = word.end - word.start
            shape = ''
            for wordIndex in range(word.start, word.end):
                shape += (' ' + doc[wordIndex].shape_)

            newWords.append([word.text,
                            0,
                            0,
                            currentSentence,
                            wordLen,
                            word.label_,
                            None,
                            None,
                            None,
                            shape])
            i = neStarts[i].end - 1
        #If not a NE, add the word if it's not a stopword or a non-alpha (not regular letters)
        else:
            if (doc[i].is_stop == False and doc[i].is_alpha == True):
                word = doc[i]

                currentSentence = getSentenceForWordPosition(i, senStarts)
                wordLen = 1

                newWords.append([word.text,
                                0,
                                0,
                                currentSentence,
```

```
                            shape])
            i = neStarts[i].end - 1
        #If not a NE, add the word if it's not a stopword or a non-alpha (not regular letters)
        else:
            if (doc[i].is_stop == False and doc[i].is_alpha == True):
                word = doc[i]

                currentSentence = getSentenceForWordPosition(i, senStarts)
                wordLen = 1

                newWords.append([word.text,
                                0,
                                0,
                                currentSentence,
                                wordLen,
                                None,
                                word.pos_,
                                word.tag_,
                                word.dep_,
                                word.shape_])
        i += 1

def oneHotEncodeColumns(df):
    columnsToEncode = ['NER', 'POS', "TAG", 'DEP']

    for column in columnsToEncode:
        one_hot = pd.get_dummies(df[column])
        one_hot = one_hot.add_prefix(column + '_')

        df = df.drop(column, axis = 1)
        df = df.join(one_hot)

    return df

```

5

STEP 5 - ANALYSING AND FINALISING THE ML COMPONENTS

As a next step we need to predict whether a word is a keyword. Here we do one-hot Encoding, Drop Unused Columns and add the missing Columns.

```python
##################################################
#    Step 6-Predict whether word is a keyword  #
##################################################

def generateDf(text):
    words = []
    addWordsForParagrapgh(words, text)

    wordColums = ['text', 'titleId', 'paragrapghId', 'sentenceId','wordCount', 'NER', 'POS', 'TAG', 'DEP','shape']
    df = pd.DataFrame(words, columns=wordColums)

    return df
def prepareDf(df):
    #One-hot encoding
    wordsDf = oneHotEncodeColumns(df)

    #Drop unused columns
    columnsToDrop = ['text', 'titleId', 'paragrapghId', 'sentenceId', 'shape']
    wordsDf = wordsDf.drop(columnsToDrop, axis = 1)

    #Add missing colums
    predictorColumns = ['wordCount','NER_CARDINAL','NER_DATE','NER_EVENT','NER_FAC','NER_GPE','NER_LANGUAGE','NER_LAW','NER_LOC','NER_MONEY','NER_NORP
    ','TAG_.','TAG_ADD','TAG_AFX','TAG_CC','TAG_CD','TAG_DT','TAG_EX','TAG_FW','TAG_IN','TAG_JJ','TAG_JJR','TAG_JJS','TAG_LS','TAG_MD','TAG_NFP','TAG_
    x','DEP_auxpass','DEP_case','DEP_cc','DEP_ccomp','DEP_compound','DEP_conj','DEP_csubj','DEP_csubjpass','DEP_dative','DEP_dep','DEP_det','DEP_dobj'

    for feature in predictorColumns:
        if feature not in wordsDf.columns:
            wordsDf[feature] = 0

    return wordsDf
def predictWords(wordsDf, df):

    predictorPickleName = vAR_Pickle_Data1
    predictor = loadPickle(predictorPickleName)

    y_pred = predictor.predict_proba(wordsDf)

    labeledAnswers = []
    for i in range(len(y_pred)):
        labeledAnswers.append({'word': df.iloc[i]['text'], 'prob': y_pred[i][0]})

    return labeledAnswers
```

6

STEP 6 - TRAIN ML

By step 6, We Extract Questions from the Potential Sentence. We also group questions and answers

```
##################################################
#           Step 7-Extract Questions            #
##################################################

vAR_Labels_Pred = vAR_model.predict(vAR_Features_Test).astype(int)
def blankAnswer(firstTokenIndex, lastTokenIndex, sentStart, sentEnd, doc):
    leftPartStart = doc[sentStart].idx
    leftPartEnd = doc[firstTokenIndex].idx
    rightPartStart = doc[lastTokenIndex].idx + len(doc[lastTokenIndex])
    rightPartEnd = doc[sentEnd - 1].idx + len(doc[sentEnd - 1])

    question = doc.text[leftPartStart:leftPartEnd] + '_____' + doc.text[rightPartStart:rightPartEnd]

    return question
```

```python
#############################################################
#           Step 8-Grouping Questions and Answers          #
#############################################################

def addQuestions(answers, text):
    doc = nlp(text)
    currAnswerIndex = 0
    qaPair = []

    #Check wheter each token is the next answer
    for sent in doc.sents:
        for token in sent:

            #If all the answers have been found, stop looking
            if currAnswerIndex >= len(answers):
                break

            #In the case where the answer is consisted of more than one token, check the following tokens as well.
            answerDoc = nlp(answers[currAnswerIndex]['word'])
            answerIsFound = True

            for j in range(len(answerDoc)):
                if token.i + j >= len(doc) or doc[token.i + j].text != answerDoc[j].text:
                    answerIsFound = False

            #If the current token is corresponding with the answer, add it
            if answerIsFound:
                question = blankAnswer(token.i, token.i + len(answerDoc) - 1, sent.start, sent.end, doc)

                qaPair.append({'question' : question, 'answer': answers[currAnswerIndex]['word'], 'prob': answers[currAnswerIndex]['prob']})

                currAnswerIndex += 1

    return qaPair
def sortAnswers(qaPairs):
    orderedQaPairs = sorted(qaPairs, key=lambda qaPair: qaPair['prob'])

    return orderedQaPairs
```

8

STEP 8 - GENERATING DISTRACTORS

Next, we do a very important step to test the knowledge. We generate very similar options to answer. This is a very crucial step in creating MCQ Questions.

```
############################################################
#           Step 9-Generationg Distractors           #
############################################################

glove_file = vAR_Training_Data1
tmp_file = vAR_Training_Data2

from gensim.scripts.glove2word2vec import glove2word2vec
glove2word2vec(glove_file, tmp_file)
model = KeyedVectors.load_word2vec_format(tmp_file)
def generate_distractors(answer, count):
    answer = str.lower(answer)

    ##Extracting closest words for the answer.
    try:
        closestWords = model.most_similar(positive=[answer], topn=count)
    except:
        #In case the word is not in the vocabulary, or other problem not loading embeddings
        return []

    #Return count many distractors
    distractors = list(map(lambda x: x[0], closestWords))[0:count]

    return distractors
def addDistractors(qaPairs, count):
    for qaPair in qaPairs:
        distractors = generate_distractors(qaPair['answer'], count)
        qaPair['distractors'] = distractors

    return qaPairs
```

9

STEP 9 - TEST ML

Yes, This is our MAIN function which actually generates Questions. Here we integrate all the functions to end our Model.

```
##############################################
#          Step 10-Main Function              #
##############################################

def generateQuestions(text, count):

    ################################
    # Extract words
    ################################
    df = generateDf(text)
    wordsDf = prepareDf(df)

    ################################
    # Predict
    ################################
    labeledAnswers = predictWords(wordsDf, df)

    ################################
    # Transform questions
    ################################
    qaPairs = addQuestions(labeledAnswers, text)

    ################################
    # Pick the best questions
    ################################
    orderedQaPairs = sortAnswers(qaPairs)

    ################################
    # Generate distractors
    ################################
    questions = addDistractors(orderedQaPairs[:count], 4)

```

9

STEP 10- VERIFY MODEL OUTCOME AND WRITE MODEL OUTCOME FOR FURTHER ANALYSIS

Yes, This is our MAIN function which actually generates Questions. Here we integrate all the functions to end our Model.

```
    ################################
    # Print
    ################################
    for i in range(count):
        options = []
        options.append(questions[i]['answer'])

        display(Markdown('### Question ' + str(i + 1) + ':'))
        print(questions[i]['question'])



        display(Markdown('#### Options:'))
        for distractor in questions[i]['distractors']:
            options.append(distractor)
#             print(distractor)

```

DeepSphere.AI
Enterprise AI and IIoT for Analytics

```
        ##################################
        # Shuffling options
        ##################################

        random.shuffle(options)
        for num,letter in enumerate(options):
            print(num+1," ",letter)

#           print(ans)
        display(Markdown('#### Answer:'))
        for x,correct in enumerate(options):
            if correct==questions[i]['answer']:
                print(x+1,correct)
        print()
f = open(vAR_Test_Data,mode='r')
vAR_Content = f.read()
print(vAR_Content)
display(Markdown('#### Content'))
print('')

generateQuestions(vAR_Content, 15)

/***********************************************************************
Disclaimer.

We are providing this code block strictly for learning and researching,this is not a
production ready code. We have no liability on this particular code under any circumstances;
Users should use this code on their own risk. All software, hardware and other products
that are referenced in these materials belong to the respective vendor who developed or who
owns this product.
/***********************************************************************
```

CONCLUSION & OUTPUT

Conclusion

We used Gaussian Naïve Bayes Model to Generate Multiple Choice Questions and similar distractors to the answer. The Model Performed well on the test data & predicted the outcome expected.

Output

Question 3:

```
Furthermore, many experts _____ AI could solve major challenges and crisis situations.
```

Options:

```
1   believed
2   know
3   believe
4   say
5   think
```

Answer:

```
3 believe
```

Question 4:

```
Furthermore, Machine learning algorithms _____ in better serving customers.
```

Options:

```
1   helps
2   to
3   helped
4   helping
5   help
```

Answer:

```
5 help
```

APPENDIX

DeepSphere.AI
Enterprise AI and IIoT for Analytics

Kubeflow Pipelines

DeepSphere.AI
Enterprise AI and IIoT for Analytics

Kubeflow Pipelines

Components and Functions

The Kubeflow Pipelines platform consists of:

- A user interface (UI) for managing and tracking experiments, jobs, and runs.
- An engine for scheduling multi-step ML workflows.
- An SDK for defining and manipulating pipelines and components.
- Notebooks for interacting with the system using the SDK.

The Kubeflow Pipelines platform has the following goals:

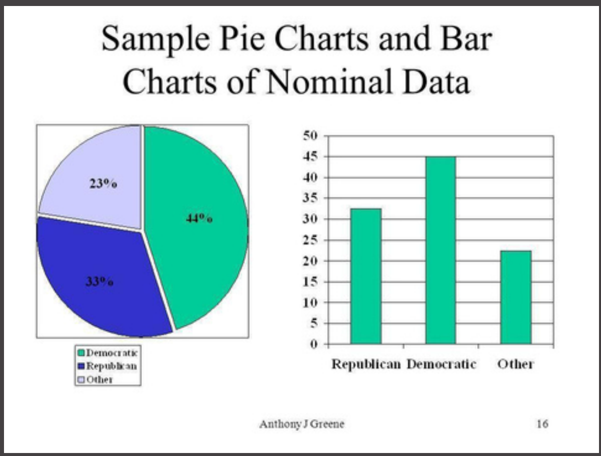
- End-to-end orchestration: enabling and simplifying the orchestration of machine learning pipelines.
- Easy experimentation: making it easy to try numerous ideas and techniques and manage various trials/experiments.
- Easy re-use: enabling to re-use components and pipelines to quickly cobble together end-to-end solutions, without having to rebuild each time.

Types of Data in Machine Learning

Types of Data in Machine Learning

Nominal Data

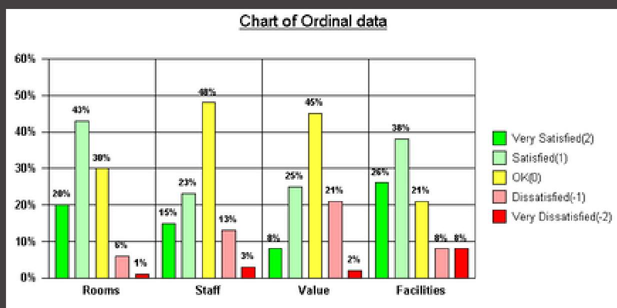
Nominal values represent discrete units and are used to label variables that have no quantitative value. Just think of them as „labels“. Note that nominal data that has no order. Therefore if you would change the order of its values, the meaning would not change. You can see two examples of nominal features below:



Types of Data in Machine Learning

Ordinal Data

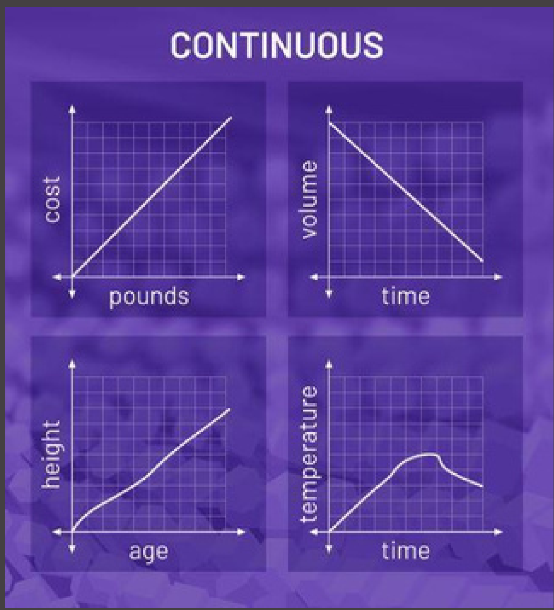
In ordinal encoding, each unique category value is assigned an integer value.For example, “red” is 1, “green” is 2, and “blue” is 3.This is called an ordinal encoding or an integer encoding and is easily reversible. Often, integer values starting at zero are used.For some variables, an ordinal encoding may be enough. The integer values have a natural ordered relationship between each other and machine learning algorithms may be able to understand and harness this relationship.



Types of Data in Machine Learning

Continuous Data

In ordinal encoding, each unique category value is assigned an integer value.For example, “red” is 1, “green” is 2, and “blue” is 3.This is called an ordinal encoding or an integer encoding and is easily reversible. Often, integer values starting at zero are used.For some variables, an ordinal encoding may be enough. The integer values have a natural ordered relationship between each other and machine learning algorithms may be able to understand and harness this relationship.

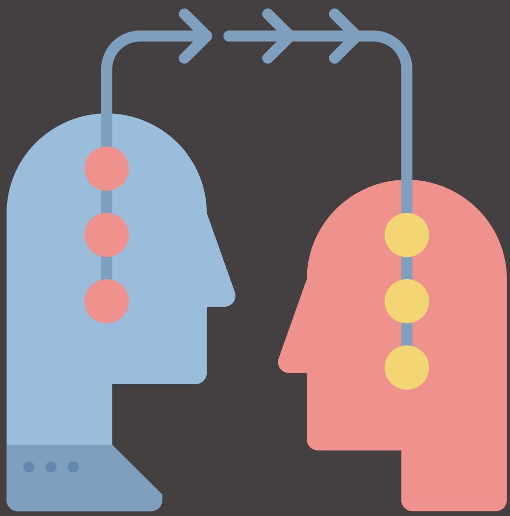


Validation in Machine Learning

Validation in Machine Learning

Validation Actual

In ordinal encoding, each unique category value is assigned an integer value.For example, “red” is 1, “green” is 2, and “blue” is 3.This is called an ordinal encoding or an integer encoding and is easily reversible. Often, integer values starting at zero are used.For some variables, an ordinal encoding may be enough. The integer values have a natural ordered relationship between each other and machine learning algorithms may be able to understand and harness this relationship.



Training

Training

Training - Actual

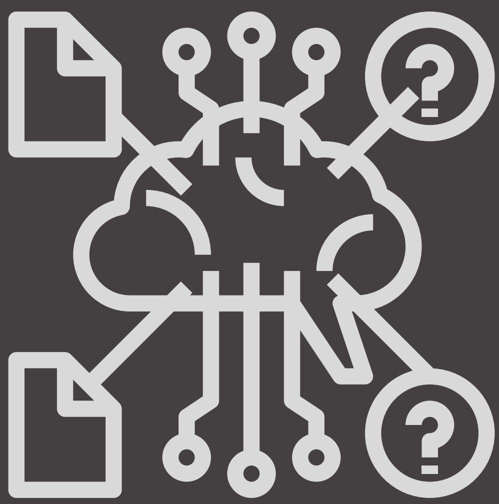
The process of training a DL model involves providing a DL algorithm (that is, the learning algorithm) with training data to learn from. The term DL model refers to the model artifact that is created by the training process. You can use the DL model to get predictions on new data for which you do not know the target.

DeepSphere.AI
Enterprise AI and IIoT for Analytics

Training

Training - Error

Training error is the error that you get when you run the trained model back on the training data. Remember that this data has already been used to train the model and this necessarily doesn't mean that the model once trained will accurately perform when applied back on the training data itself.



Prediction

DeepSphere.AI
Enterprise AI and IIoT for Analytics

Prediction

Prediction in Deep Learning

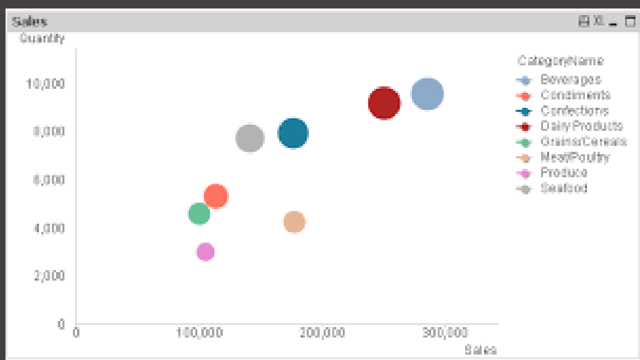
"Prediction" refers to the output of an algorithm after it has been trained on a historical dataset and applied to new data when forecasting the likelihood of a particular outcome, such as whether or not a customer will churn in 30 days. The algorithm will generate probable values for an unknown variable for each record in the new data, allowing the model builder to identify what that value will most likely be.

Data Visualisations

Data Visualisations

Bubble Chart

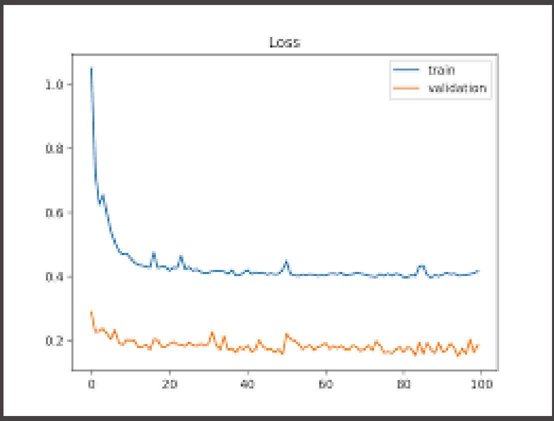
A bubble chart is a data visualization that displays multiple circles (bubbles) in a two-dimensional plot. It is a generalization of the scatter plot, replacing the dots with bubbles.



Data Visualisations

Line Chart

A line chart is, as one can imagine, a line or multiple lines showing how single, or multiple variables develop over time. It is a great tool because we can easily highlight the magnitude of change of one or more variables over a period.



DeepSphere.AI
Enterprise AI and IIoT for Analytics

Data Visualisations

Influencers

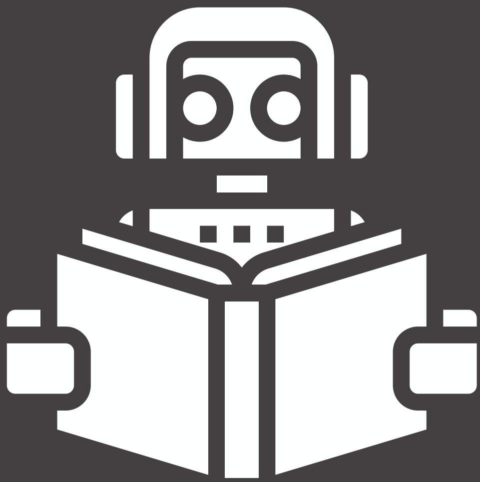
Influencers are fields that you suspect contain information about someone or something that influences or contributes to anomalies in your data.Influencers can be any field in your data. If you use data feeds, however, the field must exist in your data feed query or aggregation; otherwise it is not included in the job analysis.

DeepSphere.AI
Enterprise AI and IIoT for Analytics

Data Visualisations

Standard Deviation

Standard deviation is a number that describes how spread out the values are.A low standard deviation means that most of the numbers are close to the mean (average) value.A high standard deviation means that the values are spread out over a wider range.



Some Common ML Terms

DeepSphere.AI
Enterprise AI and IIoT for Analytics

Some Common ML Terms

“Density”

Use statistical models to find an underlying probability distribution that gives rise to the observed variables.

DeepSphere.AI
Enterprise AI and IIoT for Analytics

Some Common ML Terms

Lorenz Curve

Lorenz curve is also known under the name of "lift curve" when applied to classification/ranking. For a given range of predicted probability values, the lift represents a multiplicative increase in the positive class's rate (due to a given predictive model) over a random guess.

DeepSphere.AI
Enterprise AI and IIoT for Analytics

Some Common ML Terms

Sensitivity

Sensitivity is a measure of the proportion of actual positive cases that got predicted as positive (or true positive). This implies that there will be another proportion of actual positive cases, which would get predicted incorrectly as negative (and, thus, could also be termed as the false negative).

DeepSphere.AI
Enterprise AI and IIoT for Analytics

Some Common ML Terms

Lift

In data mining and association rule learning, lift is a measure of the performance of a targeting model (association rule) at predicting or classifying cases as having an enhanced response (with respect to the population as a whole), measured against a random choice targeting model.