ng-fundamentals-in-depth-aravindh

April 21, 2023

1.Derive log loss Gradient Descent on paper and replicate in Python

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from math import exp
from sklearn.model_selection import train_test_split
```

[2]:
```
df= pd.read_csv('data.csv')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-f4f40d0b4dbf> in <module>
----> 1 df= pd.read_csv('data_1638945638.csv')

~\anaconda3\lib\site-packages\pandas\io\parsers.py in␣
 ↪read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col,␣
 ↪usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters,␣
 ↪true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows,␣
 ↪na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates ␣
 ↪infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates,␣
 ↪iterator, chunksize, compression, thousands, decimal, lineterminator,␣
 ↪quotechar, quoting, doublequote, escapechar, comment, encoding, dialect,␣
 ↪error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map,␣
 ↪float_precision, storage_options)
    608     kwds.update(kwds_defaults)
    609
--> 610     return _read(filepath_or_buffer, kwds)
    611
    612

~\anaconda3\lib\site-packages\pandas\io\parsers.py in _read(filepath_or_buffer,␣
 ↪kwds)
    460
    461     # Create the parser.
--> 462     parser = TextFileReader(filepath_or_buffer, **kwds)
    463
    464     if chunksize or iterator:

~\anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, f, engine,␣
 ↪**kwds)
```

```
    817             self.options["has_index_names"] = kwds["has_index_names"]
    818
--> 819         self._engine = self._make_engine(self.engine)
    820
    821     def close(self):

~\anaconda3\lib\site-packages\pandas\io\parsers.py in _make_engine(self, engine
   1048             )
   1049         # error: Too many arguments for "ParserBase"
-> 1050         return mapping[engine](self.f, **self.options)  # type:␣
 ↪ignore[call-arg]
   1051
   1052     def _failover_to_python(self):

~\anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, src, **kwds)
   1865
   1866         # open handles
-> 1867         self._open_handles(src, kwds)
   1868         assert self.handles is not None
   1869         for key in ("storage_options", "encoding", "memory_map",␣
 ↪"compression"):

~\anaconda3\lib\site-packages\pandas\io\parsers.py in _open_handles(self, src,␣
 ↪kwds)
   1360         Let the readers open IOHanldes after they are done with their␣
 ↪potential raises.
   1361         """
-> 1362         self.handles = get_handle(

   1363             src,
   1364             "r",

~\anaconda3\lib\site-packages\pandas\io\common.py in get_handle(path_or_buf,␣
 ↪mode, encoding, compression, memory_map, is_text, errors, storage_options)
    640                 errors = "replace"
    641             # Encoding
--> 642             handle = open(

    643                 handle,
    644                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: 'data_1638945638.csv'
```

[ ]:
```
df
```

[ ]:
```
x1 = df.iloc[:, 2]
y1 = df.iloc[:, 4]
```

```
[ ]: x1_train,x1_test,y1_train,y1_test = train_test_split(x1 ,y1,test_size=0.
      ↪2,random_state=0)
```

```
[ ]: def normalize(x1):
         return x1 - x1.mean()
```

```
[ ]: def predict(x1, b0, b1):
         return np.array([1 / (1 + exp(-1*b0 + -1*b1*X)) for X in x1])
```

```
[ ]: def LogLoss_Gradient_descent(x1, y1,learning_rate,iterations):
         X1 = normalize(x1)
         b0 = 0
         b1 = 0
         n = len(x1)
         cost_list=[]
         for i in range(iterations):
             y_pred = predict(X1, b0, b1)
             D_b0 = 1/n*sum((y_pred-y1)) # Derivative of loss wrt b0
             D_b1 = 1/n*sum(X1 *( y_pred-y1))# Derivative of loss wrt b1
             cost = -1/n*sum(y1*np.log(y_pred)+(1-y1)*np.log(1-y_pred))
             b0 = b0 - learning_rate * D_b0         # Updating b0 and b1
             b1 = b1 - learning_rate * D_b1
             cost_list.append(cost)

             if(i%(iterations/10) == 0):
                 print("cost after ", i, "iteration is : ", cost)
         return b0, b1,cost_list
```

```
[ ]: learning_rate = 0.01
     iterations =10000
     b0,b0,cost_list = LogLoss_Gradient_descent(x1_train, y1_train,learning_rate =␣
      ↪learning_rate, iterations = iterations)
```

Cost vs Iteration

```
[ ]: plt.plot(np.arange(iterations), cost_list)
     plt.xlabel('no_of_iteration')
     plt.ylabel('cost_function')
     plt.show()
```

2.What are the different loss functions for classification

word file attached for this questions mam/sir..

3.When to use one hot encoding

We should prefer using the One Hot Encoding method when : The categorical features present in the data is not ordinal (like the countries above) When the number of categorical features present in the dataset is less so that the one-hot encoding technique can be effectively applied while building the model

4.What is dummy variable trap

Dummy Variable Trap:

The Dummy variable trap is a scenario where there are attributes that are highly correlated (Multicollinear) and one variable predicts the value of others. When we use one-hot encoding for handling the categorical data, then one dummy variable (attribute) can be predicted with the help of other dummy variables. Hence, one dummy variable is highly correlated with other dummy variables. Using all dummy variables for regression models leads to a dummy variable trap. So, the regression models should be designed to exclude one dummy variable.

For Example – Let's consider the case of gender having two values male (0 or 1) and female (1 or 0). Including both the dummy variable can cause redundancy because if a person is not male in such case that person is a female, hence, we don't need to use both the variables in regression models. This will protect us from the dummy variable trap.

5.List down the differences between linear and logistic regression

Linear Regression Logistic Regression Linear Regression is a supervised regression model. Logistic Regression is a supervised classification model. In Linear Regression, we predict the value by an integer number. In Logistic Regression, we predict the value by 1 or 0. Here no activation function is used. Here activation function is used to convert a linear regression equation to the logistic regression equation Here no threshold value is needed. Here a threshold value is added. Here we calculate Root Mean Square Error(RMSE) to predict the next weight value. Here we use precision to predict the next weight value. Here dependent variable should be numeric and the response variable is continuous to value. Here the dependent variable consists of only two categories. Logistic regression estimates the odds outcome of the dependent variable given a set of quantitative or categorical independent variables. It is based on the least square estimation. It is based on maximum likelihood estimation. Here when we plot the training datasets, a straight line can be drawn that touches maximum plots. Any change in the coefficient leads to a change in both the direction and the steepness of the logistic function. It means positive slopes result in an S-shaped curve and negative slopes result in a Z-shaped curve. Linear regression is used to estimate the dependent variable in case of a change in independent variables. For example, predict the price of houses. Whereas logistic regression is used to calculate the probability of an event. For example, classify if tissue is benign or malignant. Linear regression assumes the normal or gaussian distribution of the dependent variable. Logistic regression assumes the binomial distribution of the dependent variable.

```
[ ]:
```