LAPORAN TUGAS BESAR CLASSIFICATION (SUPERVISED LEARNING)
MATA KULIAH PEMBELAJARAN MESIN

Disusun oleh :

Imam Rafiif Arrazaan (1301194152) IF4307

Arvinda Dwi Safira (1301190083) IF4307

PROGRAM STUDI S1 INFORMATIKA

FAKULTAS INFORMATIKA

UNIVERSITAS TELKOM BANDUNG

BANDUNG

2021

DAFTAR ISI

1. PENDAHULUAN

Laporan ini dibuat untuk memenuhi Tugas Besar Mata Kuliah Pembelajaran Mesin tahap kedua. Tahap kedua dilakukan secara berkelompok, yaitu Classification (Supervised Learning) memprediksi apakah pelanggan tertarik untuk membeli kendaraan baru atau tidak berdasarkan data pelanggan di dealer.

Untuk menyelesaikan tugas ini, kami akan melakukan perbandingan hasil dari 2 algoritma Supervised Learning, yaitu Naive Bayes dan Logistic Regression.

2. FORMULASI MASALAH

Dataset yang digunakan untuk melakukan tugas ini adalah “kendaraan_train.csv” dan “kendaraan_test.csv” yang berisi data pelanggan di dealer dengan 12 kolom data. Hal yang perlu dilakukan dalam tugas ini adalah dengan menggunakan classification (supervised learning), mahasiswa ditugaskan untuk memprediksi apakah pelanggan tertarik untuk membeli kendaraan baru atau tidak berdasarkan data pelanggan di dealer.

3. EKSPLORASI DAN PERISAPAN DATA

Sebelum menjalankan algoritma Supervised Learning yang telah ditentukan, dataset perlu dipersiapkan terlebih dahulu agar mendapatkan hasil yang baik dan tepat. Untuk itu, akan dilakukan beberapa tahapan pada kedua dataset.

3.1 Import Data dan Library

Dalam membangun program membutuhkan beberapa library utama seperti pandas, dan juga beberapa library pendukung yang nantinya digunakan untuk membangun program.

```
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np
import gdown
```

Gambar 3.1.1 Import Library

3.2 Read Data

Memasukkan dataset ke dalam dataframe dan melihat isi dari dataframe yang telah dibuat.

```
df = pd.read_csv('kendaraan_train.csv')
```

Gambar 3.2.1 Membaca Data

3.3 Basic Information

Menampilkan informasi mengenai dataset, diantaranya jumlah data, nama kolom, data yang tidak kosong dan tipe data.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 285831 entries, 0 to 285830
Data columns (total 12 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   id                 285831 non-null  int64  
 1   Jenis_Kelamin      271391 non-null  object 
 2   Umur               271617 non-null  float64
 3   SIM                271427 non-null  float64
 4   Kode_Daerah        271525 non-null  float64
 5   Sudah_Asuransi     271602 non-null  float64
 6   Umur_Kendaraan     271556 non-null  object 
 7   Kendaraan_Rusak    271643 non-null  object 
 8   Premi              271262 non-null  float64
 9   Kanal_Penjualan    271532 non-null  float64
 10  Lama_Berlangganan  271839 non-null  float64
 11  Tertarik           285831 non-null  int64  
dtypes: float64(7), int64(2), object(3)
memory usage: 26.2+ MB
```

Gambar 3.3.1 Basic Information

3.4 Menangani Missing Value, Duplicated Data, dan Outlier

Perlu dilakukan penanganan dataset sebelum menggunakan data, seperti missing value, data yang terduplikasi, dan outlier/pencilan.

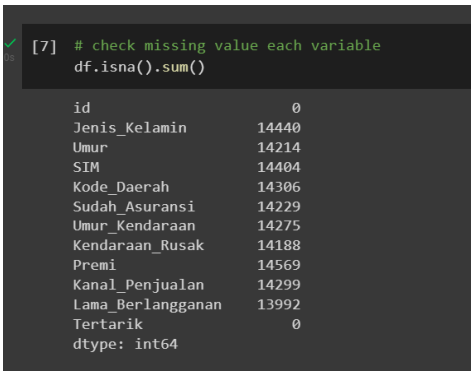
Pertama akan dilakukan pengecekan pada duplicated data dan missing value. Seperti pada Gambar 3.4.1, terdapat 0 duplicated data yang artinya tidak ada data yang terduplikasi. Namun terdapat 142916 missing value pada dataset yang perlu ditangani.

```
[6]  # check duplicated data and missing value
     print(df.duplicated().sum())
     print(df.isna().sum().sum())

     0
     142916
```

Gambar 3.4.1 Jumlah Keseluruhan Duplicated Data dan Missing Value

Penanganan missing value dilakukan dengan 2 cara, yaitu drop data dan mengisi baris missing value dengan rata-rata kolom tersebut. Pada Gambar 3.4.2 memeriksa berapa banyak null yang ada pada tiap kolom.

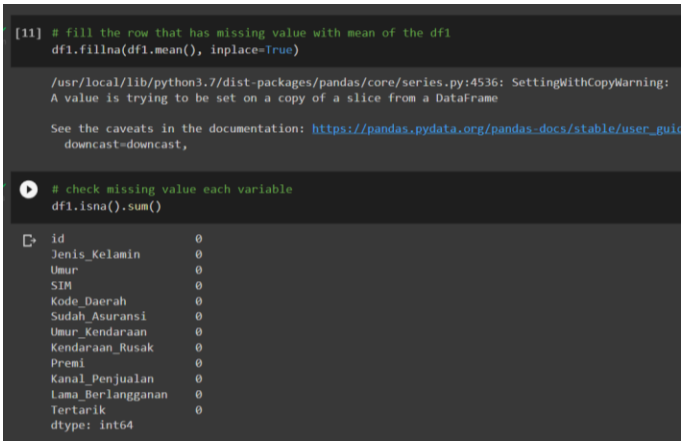


Gambar 3.4.2 Jumlah Missing Value tiap kolom

Kemudian drop missing value pada seluruh kolom kecuali Umur, Premi, dan Lama_Berlangganan seperti pada Gambar 3.4.3, karena ketiga variabel tersebut dapat dimasukkan dengan rata-rata data tersebut.

Gambar 3.4.3 Drop baris yang berisi missing value

Missing value pada ketiga kolom tersebut tidak didrop, melainkan diisi dengan data rata-rata dari masing-masing data kolom tersebut, yang ditunjukkan pada Gambar 3.4.4. Sehingga missing value menjadi 0 atau tidak ada missing value lagi.



Gambar 3.4.4 Mengisi missing value dengan nilai rata-rata

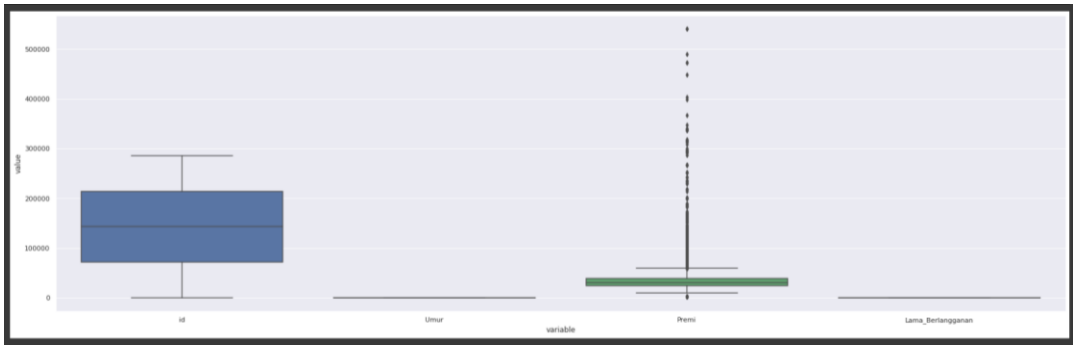
Terakhir akan memeriksa dan menangani outlier, sebelumnya data dimasukkan ke dalam dataframe baru yaitu df2 yang berisi kolom 'id', 'Umur', 'Premi', dan 'Lama_Berlangganan'. Setelah itu, kita bisa memeriksa apakah ada outlier pada dataset atau tidak. Untuk memeriksa outlier akan menggunakan box plot. Hasil dari pengecekan outlier dapat dilihat pada Gambar 3.4.6.

```
# df2 with id, Umur, and Lama_berlangganan from df1
df2 = df1[['id', 'Umur', 'Premi', 'Lama_Berlangganan']]

# check outliers
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(rc={'figure.figsize':(30,9)})
sns.boxplot(x="variable", y="value", data=pd.melt(df2))

plt.show()
```

Gambar 3.4.5 Memeriksa outlier



Gambar 3.4.6 Hasil Box Plot Outlier

Terlihat pada Gambar 3.4.6, kolom “Premi” memliki banyak outlier, dan outlier tersebut perlu ditangani. Maka dilakukan perhitungan menggunakan zscore seperti di Gambar 3.4.7.

```
# handling outliers
from scipy import stats
z = np.abs(stats.zscore(df2))
print(z)
```

```
[[1.73388675e+00 5.82560092e-01 1.48922090e-01 7.01190590e-01]
 [1.73387463e+00 6.05947615e-01 2.82398908e-01 4.48452533e-02]
 [1.73385037e+00 1.26622967e+00 1.66986319e+00 1.11701385e+00]
 ...
 [1.73283688e+00 1.04475753e+00 1.15183316e+00 8.76491768e-01]
 [1.73284901e+00 1.92651173e+00 7.74199091e-04 1.41461598e+00]
 [1.73286114e+00 4.07862997e-01 3.57140086e-01 1.34938567e+00]]
```

```
threshold = 3
print(np.where(z > 3))
```

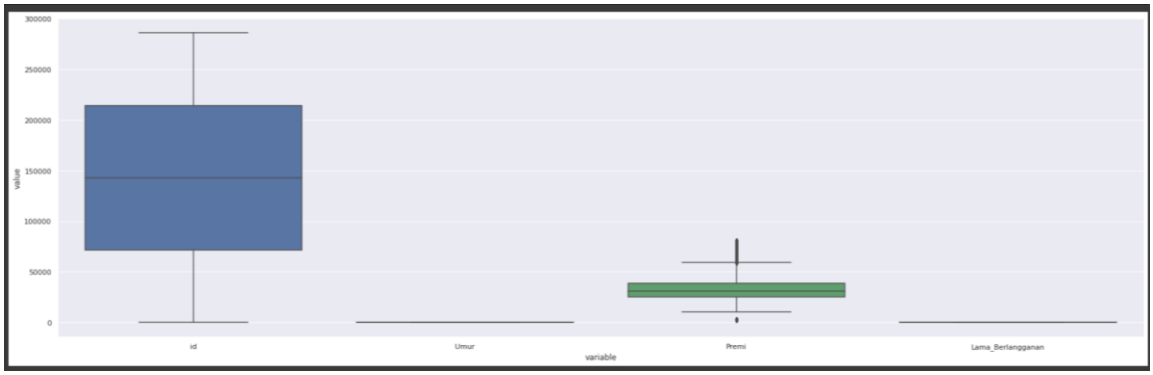
```
(array([  530,  1110,  1621, ..., 199207, 199292, 199340]), array([2, 2, 2, ..., 2, 2, 2]))
```

```
df3 = df2[(z < 3).all(axis=1)]
```

```
print(df2.shape)
print(df3.shape)
```

Gambar 3.4.7 Handling Outlier Premi

Hasil dari handling outlier dapat dilihat pada Gambar 3.4.8. Sesuai yang ada pada gambar tersebut, outlier berkurang namun tidak sepenuhnya hilang.



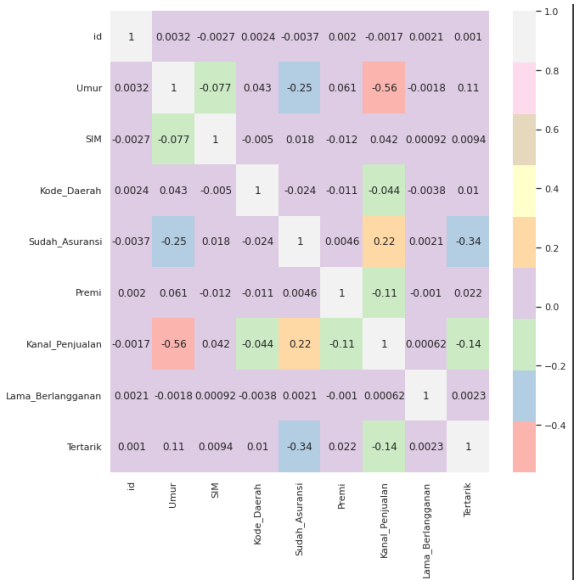
Gambar 3.4.8 Kolom Premi setelah handling outlier

3.5 Memeriksa Korelasi Antar Variabel

Untuk menjalankan algoritma Naive Bayes memerlukan 3 kolom data. Maka dari itu perlu ditentukan kolom data mana yang akan digunakan dengan cara melihat korelasi antar variabel yang ada pada dataset. Untuk memeriksa korelasi antar variabel akan menggunakan heatmap, seperti pada Gambar 3.5.1.

```
# check correlation between variables
import seaborn as sns
sns.set(rc={'figure.figsize':(15,15)})
def heatmap(data):
  sns.heatmap(data.corr(), vmax=1, annot=True, cmap='Pastel1')
```

Gambar 3.5.1 Cek korelasi antar variabel



Gambar 3.5.2 Hasil korelasi antar variabel

Berdasarkan hasil dari korelasi antar variabel yang ditunjukkan dengan heatmap pada Gambar 3.5.2, maka kami memutuskan untuk menggunakan

variabel “Sudah_Asuransi” karena memiliki nilai korelasi tertinggi dengan variabel Tertarik. Selain itu, kami juga memilih variabel “Kendaraan_Rusak” dan “Umur_Kendaraan” karena ketiga variabel tersebut memiliki variansi nilai yang tidak banyak sehingga memudahkan dalam mengeksekusi code from scratch.

Selanjutnya akan dilakukan 4 variabel yang akan dimasukkan ke dalam dataframe baru yaitu dfn, seperti pada Gambar 3.5.3.

```
#df2 = df2.drop(['Tertarik'], axis=1)
dfn = df1[['Sudah_Asuransi','Kendaraan_Rusak', 'Umur_Kendaraan', 'Tertarik']]
dfn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199438 entries, 0 to 285830
Data columns (total 4 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   Sudah_Asuransi   199438 non-null  float64
 1   Kendaraan_Rusak  199438 non-null  object
 2   Umur_Kendaraan   199438 non-null  object
 3   Tertarik         199438 non-null  int64
dtypes: float64(1), int64(1), object(2)
memory usage: 7.6+ MB
```

Gambar 3.5.3 Dataset final tanpa missing value

3.6 Mengubah Tipe Data

Selanjutnya perlu menyamakan tipe data variabel agar dapat digunakan khususnya untuk algoritma Naive Bayes.

```
#df2 = df2.drop(['Tertarik'], axis=1)
dfn = df1[['Sudah_Asuransi','Kendaraan_Rusak', 'Umur_Kendaraan', 'Tertarik']]
dfn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199438 entries, 0 to 285830
Data columns (total 4 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   Sudah_Asuransi   199438 non-null  float64
 1   Kendaraan_Rusak  199438 non-null  object
 2   Umur_Kendaraan   199438 non-null  object
 3   Tertarik         199438 non-null  int64
dtypes: float64(1), int64(1), object(2)
memory usage: 7.6+ MB
```

Gambar 3.6.1 Data yang akan digunakan

Gamber 3.6.1 merupakan tipe data variabel sebelum dibuah, kemudian pada Gambar 3.6.2 merupakah hasil tipe data setelah diubah..

```
dfn = dfn.astype(str)
dfn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199438 entries, 0 to 285830
Data columns (total 4 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   Sudah_Asuransi   199438 non-null  object
 1   Kendaraan_Rusak  199438 non-null  object
 2   Umur_Kendaraan   199438 non-null  object
 3   Tertarik         199438 non-null  object
dtypes: object(4)
memory usage: 7.6+ MB
```

Gambar 3.6.2 Data setelah disamakan tipe datanya

3.7 Bag of Words

Selanjutnya pada dataset “kendaraan_train.csv”, akan diterapkan teknik bag of word. Pada kolom “Tertarik” akan digunakan sebagai label, berdasarkan 3 kolom lainnya, akan dikumpulkan data-data di kolom yang memiliki hasil “Tertarik” = 1 dan seperti pada Gambar 3.7.1.

```
# berdasarkan 3 kolom, mengumpulkan data2 di kolom yang hasil tertariknya = 1
str_tertarik = ''
str_tidakTertarik = ''

n_true = 0
n_false = 0

for _, data in dfn.iterrows():
  if data['Tertarik']=="1":
    str_tertarik += data['Sudah_Asuransi'] + '_' + data['Kendaraan_Rusak'] + '_' + data['Umur_Kendaraan'] + '_'
    n_true += 1
  else:
    str_tidakTertarik += data['Sudah_Asuransi'] + '_' + data['Kendaraan_Rusak'] + '_' + data['Umur_Kendaraan'] + '_'
    n_false += 1
```

Gambar 3.7.1 Bag of words 1

Kemudian menampung setiap data dengan teknik bag of word (bow), yang membedakan “Tertarik” = 1 sebagai bow_tertarik, dan “Tertarik” = 0 sebagai bow_tidakTertarik.

```
V = len(set((str_tertarik + ' ' + str_tidakTertarik).split()))

# menampung setiap data yang tertarik = 1
bow_tertarik = {}
bow_tidakTertarik = {}

for word in str_tertarik.split('_'):
  if word in bow_tertarik.keys():
    bow_tertarik[word] += 1
  else:
    bow_tertarik[word] = 1

for word in str_tidakTertarik.split('_'):
  if word in bow_tidakTertarik.keys():
    bow_tidakTertarik[word] += 1
  else:
    bow_tidakTertarik[word] = 1
```

Gambar 3.7.2 Bag of words 2

Kemudian dihitung juga probabilitas bag of word (bow) seperti pada Gambar 3.7.3.

```
# menghitung probabilitas bag of word
bow_tertarik_prob = {}
bow_tidakTertarik_prob = {}

for key in bow_tertarik.keys():
  bow_tertarik_prob[key] = bow_tertarik[key] / sum_bow_tertarik

for key in bow_tidakTertarik.keys():
  bow_tidakTertarik_prob[key] = bow_tidakTertarik[key] / sum_bow_tidakTertarik
```

Gambar 3.6.7 Bag of words 3

3.8 Mengubah Value Data Menjadi Angka

Untuk algoritma Logistic Regression, dataset yang masih berupa string perlu diubah menjadi angka. Maka akan digunakan Label Encoder untuk mengubah data string menjadi angka yang sesuai dengan data asli.

Beberapa variabel yang perlu diubah valuenya pada data Train adalah "Umur_Kendaraan", dan "Kendaraan_Rusak", seperti yang dapat dilihat pada Gambar 3.8.1.

```
# mengubah data string menjadi angka sesuai data
labelencoder = LabelEncoder()
df_train_LGn['Umur_Kendaraan'] = labelencoder.fit_transform(df_train_LGn['Umur_Kendaraan'])
df_train_LGn['Kendaraan_Rusak'] = labelencoder.fit_transform(df_train_LGn['Kendaraan_Rusak'])
```

Gambar 3.8.1 Label Encoder Data Train

Kemudian juga dilakukan pada data Test untuk variabel "Umur_Kendaraan", "Jenis_Kelamin", dan "Kendaraan_Rusak" seperti Gambar 3.8.2.

```
labelencoder = LabelEncoder()
df_test_LG['Umur_Kendaraan'] = labelencoder.fit_transform(df_test_LG['Umur_Kendaraan'])

df_test_LG['Jenis_Kelamin'] = labelencoder.fit_transform(df_test_LG['Jenis_Kelamin'])
df_test_LG['Kendaraan_Rusak'] = labelencoder.fit_transform(df_test_LG['Kendaraan_Rusak'])
```

Gambar 3.8.1 Label Encoder Data Test

3.9 Normalisasi

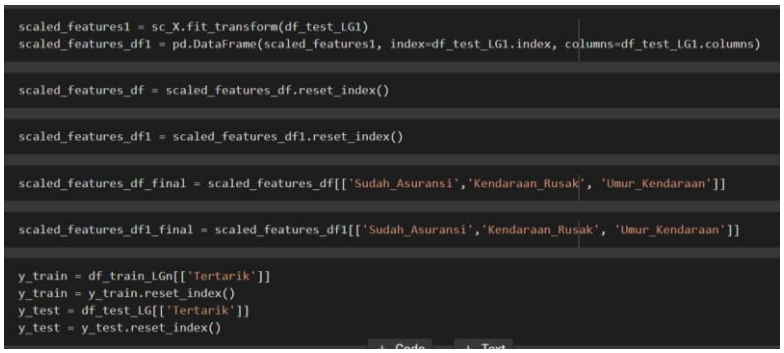
Untuk memudahkan dalam mengeksekusi Logistic Regression, data perlu dinormalisasi terlebih dahulu agar range nilainya tidak terlalu besar. Dengan menggunakan Standard Scaler, normalisasi dilakukan pada data Train seperti pada Gambar 3.9.1.

```
# normalisasi agar range tidak terlalu besar
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
scaled_features = sc_X.fit_transform(df_train_LGn)
scaled_features_df = pd.DataFrame(scaled_features, index=df_train_LGn.index, columns=df_train_LGn.columns)
```

Gambar 3.9.1 Label Encoder

3.10 Scaling

Scaling dilakukan pada data Test yang akan digunakan untuk algoritma Logistic Regression, scaling dilakukan seperti pada Gambar 3.10.1.



Gambar 3.10.1 Scaling

4. PEMODELAN

Pada tahap pemodelan menggunakan Naive Bayes dan Logistic Regression. Naive Bayes merupakan sebuah metode penggolongan berdasarkan probabilitas sederhana dan dirancang untuk dipergunakan dengan asumsi bahwa antar satu kelas dengan kelas yang lain tidak saling tergantung (independen). Sedangkan Logistic Regression merupakan suatu metode analisis statistika yang mendeskripsikan hubungan antara peubah respon (dependent variable) yang bersifat kualitatif memiliki dua kategori atau lebih dengan satu atau lebih peubah penjelas (independent variable) berskala kategori atau interval.

4.1 Naive Bayes

Dalam mengeksekusi dataset dengan algoritma Naive Bayes, terdapat beberapa fungsi yang diperlukan, yang pertama adalah fungsi untuk memprediksi kemungkinan kasus seperti pada Gambar 4.1.1.

```
# Function for predicting the possible cases
def pred(sentence):
  res_true = 1
  res_false = 1

  for word in sentence.split():
    if word in bow_tertarik.keys():
      res_true *= bow_tertarik_prob[word]
    else:
      res_true *= 1 / sum_bow_tertarik

  for word in sentence.split():
    if word in bow_tidakTertarik.keys():
      res_false *= bow_tidakTertarik_prob[word]
    else:
      res_false *= 1 / sum_bow_tidakTertarik

  res_true *= n_true / (n_true + n_false)
  res_false *= n_false / (n_true + n_false)

  if res_true > res_false:
    return 1
  else:
    return 0
```

Gambar 4.1.1 Fungsi untuk memprediksi kemungkinan kasus

Kemudian dibuat list untuk masing-masing variabel yang berisi prediksi data Test.

```
# Prediction on test set
list1 = list(df1_test['Sudah_Asuransi'])
list2 = list(df1_test['Kendaraan_Rusak'])
list3 = list(df1_test['Umur_Kendaraan'])

res = list(zip(list1, list2, list3))

# printing result
# print ("All possible permutations are : " +  str(res))
```

Gambar 4.1.2 List berisi prediksi data Test

Kemudian terdapat fungsi yang digunakan untuk membuat list berisi String dari list yang telah dibuat pada Gambar 4.1.2. List berisi String tersebut akan dimasukkan ke dalam variabel dat seperti pada Gambar 4.1.3.

```
def listToString(s):

    # initialize an empty string
    str1 = " "

    # return string
    return (str1.join(s))

# Driver code
dat = []*len(res)
for i in range(len(res)):
  dat.append(listToString(res[i]))
```

Gambar 4.1.3 List berisi prediksi data Test

Variabel dat tersebut berisi value “Umur_Kendaraan”, “Jenis_Kelamin”, dan “Kendaraan_Rusak” yang telah digabung. Selanjutnya dat akan dimasukkan ke dalam dataframe baru dengan nama test dan kolom ‘Spec’.

Kemudian akan dibuat list baru bernama hasil yang merupakan hasil “Tertarik” dari list dat pada fungsi pred (fungsi untuk mendapatkan prediksi) untuk menentukan hasil tertarik atau tidak berdasarkan ketika data variabel tersebut. Untuk memudahkan dalam melihat hasil dari fungsi tersebut, list hasil kemudian diappend ke dataframe ‘test’ yang sebelumnya telah berisi kolom ‘Spec’ yang dapat dilihat pada Gambar 4.1.4.

```
hasil = []
for i in range(len(dat)):
  hasil.append(pred(dat[i]))
```

```
test['Tertarik'] = hasil
test.head()
```

	Spec	Tertarik
0	0.0 Pernah 1-2 Tahun	1
1	1.0 Tidak < 1 Tahun	0
2	1.0 Tidak < 1 Tahun	0
3	1.0 Tidak 1-2 Tahun	0
4	0.0 Pernah 1-2 Tahun	1

Gambar 4.1.4 Hasil prediksi

Kemudian untuk memeriksa tingkat keakuratan dari hasil tersebut, akurasi akan diperiksa dengan menggunakan nilai “Tertarik” dari dataset asli.

```
a = test['Tertarik']
b = df_test['Tertarik']
```

```
print("Akurasi: ", end="")
sum(1 for x,y in zip(a,b) if x == y) / float(len(a))
```

```
Accuracy: 0.6104872058607443
```

Gambar 4.1.5 Akurasi Naive Bayes

Berdasarkan hasil perhitungan akurasi dari variabel “Tertarik” yang baru pada Gambar 4.1.5, didapatkan tingkat akurasinya adalah 0.610.

4.2 Logistic Regression

Algoritma Supervised Learning selanjutnya adalah Logistic Regression. Algoritma Logistic Regression yang memiliki beberapa fungsi yang dibutuhkan.

Fungsi pertama yaitu fungsi fit yang digunakan untuk train model. Pada fungsi tersebut akan inisialisasi number of features dan number of training examples, juga inisialisasi weight. Variabel b pada inisialisasi weight merupakan bias (parameter). Lalu ada juga algoritma Gradient Descent untuk mencari nilai optimal dari parameter yang ada. Fungsi untuk train model dapat dilihat pada Gambar 4.2.1.

```
# Function for model training
def fit(self, X, Y):
  # no of trainin examples, no of features
  self.m, self.n = X.shape

  # weight initialization
  self.W = np.zeros(self.n)
  self.b = 0
  self.X = X
  self.Y = Y

  # gradient descent learning
  for i in range(self.iterations) :
    self.update_weights()
  return self
```

Gambar 4.2.1 Fungsi utama Logistic Regression

Untuk menjalankan algoritma Gradient Descent, diperlukan beberapa fungsi pembantu yang akan menghitung gradient dan memperbarui nilai weight seperti pada Gambar 4.2.2.

```
# Helper function to update weights in gradient descent
def update_weights(self) :
  A = 1/(1+np.exp(-(self.X.dot(self.W)+self.b)))

  # calculate gradients
  tmp = (A-self.Y.T)
  tmp = np.reshape(tmp,self.m)
  dW = np.dot(self.X.T,tmp)/self.m
  db = np.sum(tmp)/self.m

  # update weights
  self.W = self.W - self.learning_rate * dW
  self.b = self.b - self.learning_rate * db

  return self
```

Gambar 4.2.2 Fungsi pembantu Logistic Regression

Kemudian ada pula fungsi predict atau hypothetical function pada Gambar 4.2.3, yang digunakan untuk menghitung hipotesis atau prediksi dengan angka 1 atau 0.

```
# Hypothetical function h(x)
def predict(self, X) :
  Z = 1/(1+np.exp(-(X.dot(self.W)+self.b)))
  Y = np.where(Z>0.5, 1, 0)
  return Y
```

Gambar 4.2.3 Fungsi pembantu Logistic Regression

Terakhir adalah untuk menjalankan seluruh fungsi yang telah dibuat sebelumnya, pertama akan memasukkan dataset yang telah dipersiapkan dan menjalankan model train seperti pada Gambar 4.2.4.

```
# Driver code

# Assigning dataset into train and test set
X_train = scaled_features_df_final
X_test = scaled_features_df1_final
Y_train = y_train['Tertarik']
Y_test = y_test['Tertarik']

# Model training
model = LogisticRegression(learning_rate = 0.01, iterations = 1000)
model.fit(X_train, Y_train)

# Prediction on test set
Y_pred = model.predict(X_test)
```

Gambar 4.2.4 Driver code

Hasil dari prediksi berapa pada variabel Y_pred.

```
[78] Y_pred

     array([0, 0, 0, ..., 0, 0, 0])
```

Gambar 4.2.5 Hasil prediksi

Tingkat akurasi dari penggunaan algoritma Logistic Regression adalah 0.876.

```
a = Y_pred
b = df_test['Tertarik']
print("Accuracy: ", end="")
sum(1 for x,y in zip(a,b) if x == y) / float(len(a))
```

```
Accuracy: 0.8769705493398267
```

Gambar 4.2.6 Akurasi Logistic Regression

5. KESIMPULAN

Berdasarkan percobaan yang kami lakukan menggunakan dua model klasifikasi yang berbeda, dapat diambil kesimpulan, yaitu:

1. Dataset yang disediakan merupakan data yang kurang berkualitas. Hal itu disebabkan data yang ada memiliki sangat banyak nilai yang hilang. Hampir setengah dari jumlah data memiliki nilai yang hilang dari berbagai atribut yang ada.
2. Ketika tidak dilakukan optimasi dalam algoritma model yang dibuat, hal tersebut dapat mengakibatkan bertambahnya waktu yang diperlukan untuk menjalankan program.
3. Model klasifikasi Naive Bayes kurang cocok digunakan terhadap dataset yang diberikan. Hal ini diakibatkan algoritma Naive Bayes yang hanya melakukan perbandingan terhadap peluang kemunculan setiap data.
4. Model klasifikasi Logistic Regression memiliki akurasi yang lebih besar dibandingkan dengan model klasifikasi Naive Bayes untuk pengklasifikaisan dataset yang diberikan dengan selisih akurasi lebih dari 20%.

6. TAUTAN

Source Code :
https://colab.research.google.com/drive/1_pcH4FJnd5UtkFxSbGYcvII5FyQt_k0w?usp=sharing