# LAPORAN TUGAS BESAR 2 KLASIFIKASI

dibuat untuk memenuhi tugas 2 Pembelajaran Mesin


Oleh:

Nur Laili Ramadhani (1301194098)
Fauzi Arya Surya Abadi (1301194101)

CII3C3-IF-43-01

Tahun Ajaran 2021/2022

Program Studi S1 Informatika
Fakultas Informatika
Universitas Telkom
Bandung

# Daftar Isi

## 1. Pendahuluan

Mata Kuliah Pembelajaran Mesin atau bisa disebut juga dengan Machine Learning merupakan mata kuliah wajib yang diambil pada semester 5. Di mata kuliah ini, terdapat 2 tugas besar. Pada tugas besar pertama, mahasiswa diminta menentukan Clustering yang dilakukan secara individu dan tugas besar kedua yaitu mahasiswa diminta menentukan klasifikasi dan dikerjakan secara berkelompok yang terdiri dari dua orang di setiap kelompok. Pada tugas kedua ini, mahasiswa diminta untuk mengklasifikasi dataset yang telah disediakan. Dataset yang digunakan ada 2 yaitu, *kendaraan_train.csv* dan *kendaraan_test.csv*. Di dalam dataset *kendaraan_train* terdapat 12 kolom yaitu: *id, jenis_kelamin, umur, SIM, kode_daerah, sudah_asuransi, umur_kendaraan, kendaraan_rusak, premi, kanal_penjualan, lama_berlangganan, tertarik.* sedangkan pada dataset kendaraan_test terdapat 11 kolom yaitu: *jenis_kelamin, umur, SIM, kode_daerah, sudah_asuransi, umur_kendaraan, kendaraan_rusak, premi, kanal_penjualan, lama_berlangganan.*

Pada tugas klasifikasi ini kami menggunakan 2 model *algoritma Supervised Learning (SL)* yaitu *Logistic Regression* dan *Linear Regression* yang dibangun, serta menggunakan library dari kedua model tersebut untuk diketahui model mana yang terbaik untuk pengklasifikasian terhadap dataset yang telah ditentukan. Bahasa pemrograman yang digunakan yaitu Python. Pada tugas klasifikasi ini, kami menggunakan tools untuk mengerjakannya yaitu dengan *Jupyter Notebook* pada Google Collab.

## 2. Formulasi Masalah

Dalam tugas clustering ini kami menggunakan file dari dataset yang bernama kendaraan_train.csv dan kendaraan_test.csv. Data tersebut memiliki beberapa atribut *Jenis_Kelamin, Umur, SIM, Kode_Daerah, Sudah_Asuransi, Umur_Kendaraan, Kendaraan_Rusak, Premi, Kanal_Penjualan, Lama_Berlangganan, Tertarik.* Jumlah data didalam dataset tersebut berjumlah 285.831 records. Didalam data tersebut pastinya terdapat data kosong atau missing value, dan ada beberapa data yang bertipe kategorikal dari suatu atribut diatas. Namun berbeda dengan data set *Kendaraan_Test,* data set tersebut tidak memiliki missing value. Selain missing value atau nilai kosong, pada kedua data set tersebut juga memiliki nilai non-numeric atau berkategori pada beberapa atribut di dalamnya, maka perlu pelabelan pada data

tersebut karena bila terdapat nilai katagorial non-numeric maka dapat mempengaruhi proses pengklasifikasian.

# 3. Eksplorasi dan Persiapan Data

## 3.1 Membaca Data Set

- Data set kendaraan_train.csv

```
# Load data kendaraan train
df_train = pd.read_csv('https://github.com/fasa2297/fml-ml2-classification/blob/main/kendaraan_train.csv?raw=true')
df_train.head()
```

| | id | Jenis_Kelamin | Umur | SIM | Kode_Daerah | Sudah_Asuransi | Umur_Kendaraan | Kendaraan_Rusak | Premi | Kanal_Penjualan | Lama_Berlangganan | Tertarik |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | Wanita | 30.0 | 1.0 | 33.0 | 1.0 | < 1 Tahun | Tidak | 28029.0 | 152.0 | 97.0 | 0 |
| 1 | 2 | Pria | 48.0 | 1.0 | 39.0 | 0.0 | > 2 Tahun | Pernah | 25800.0 | 29.0 | 158.0 | 0 |
| 2 | 3 | NaN | 21.0 | 1.0 | 46.0 | 1.0 | < 1 Tahun | Tidak | 32733.0 | 160.0 | 119.0 | 0 |
| 3 | 4 | Wanita | 58.0 | 1.0 | 48.0 | 0.0 | 1-2 Tahun | Tidak | 2630.0 | 124.0 | 63.0 | 0 |
| 4 | 5 | Pria | 50.0 | 1.0 | 35.0 | 0.0 | > 2 Tahun | NaN | 34857.0 | 88.0 | 194.0 | 0 |

Dari dataset kendaraan_train.csv, memiliki data berjumlah 285.831 records.danmasing-masing atribut yang memiliki missing value atau nilai kosong dan bertipe kategorikal.

- Data set kendaraan_test.csv

```
# Load data kendaraan test
df_test = pd.read_csv('https://github.com/fasa2297/fml-ml2-classification/blob/main/kendaraan_test.csv?raw=true')
df_test.head()
```

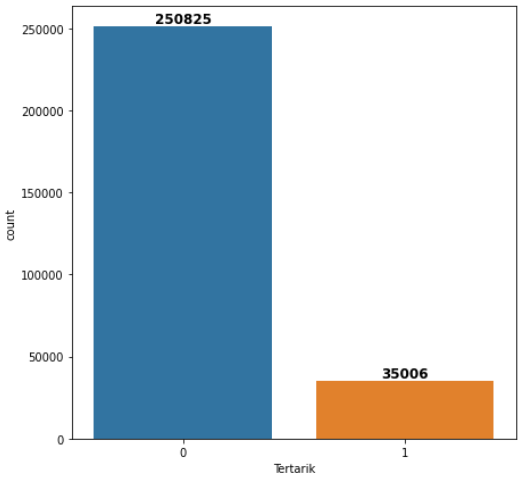
| | Jenis_Kelamin | Umur | SIM | Kode_Daerah | Sudah_Asuransi | Umur_Kendaraan | Kendaraan_Rusak | Premi | Kanal_Penjualan | Lama_Berlangganan | Tertarik |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Wanita | 49 | 1 | 8 | 0 | 1-2 Tahun | Pernah | 46963 | 26 | 145 | 0 |
| 1 | Pria | 22 | 1 | 47 | 1 | < 1 Tahun | Tidak | 39624 | 152 | 241 | 0 |
| 2 | Pria | 24 | 1 | 28 | 1 | < 1 Tahun | Tidak | 110479 | 152 | 62 | 0 |
| 3 | Pria | 46 | 1 | 8 | 1 | 1-2 Tahun | Tidak | 36266 | 124 | 34 | 0 |
| 4 | Pria | 35 | 1 | 23 | 0 | 1-2 Tahun | Pernah | 26963 | 152 | 229 | 0 |

Dari dataset kendaraan_train.csv, memiliki data berjumlah 47.639 records.dan masing-masing atribut yang memiliki missing value atau nilai kosong. Semua sumber data set kami letakan pada repository github.

## 3.2 Cek nilai Ketertarikan

```
plt.figure(figsize=(7,7))
plots = sns.countplot(x="Tertarik", data=df_train); rects = plots.patches
labels = [df_train["Tertarik"].value_counts()[0], df_train["Tertarik"].value_counts()[1]]
for rect, label in zip(rects, labels):
  height = rect.get_height()
  plt.text(rect.get_x() + rect.get_width() / 2, height, label,ha='center', va='bottom',fontweight=600, fontsize=12)
plt.show()
```

Dalam diagram batang dibawah ini merupakan representasi dari jumlah orang yang tertarik pada atribut “Tertarik” pada kendaraan_train dengan jumlah data sebesar 250825 dan yang tidak tertarik sebesar 35006.




### 3.3 Cek missing values

```
1 # Melihat nilai kosong di data training
2 df_train.isnull().sum()
```

```
id                      0
Jenis_Kelamin       14440
Umur                14214
SIM                 14404
Kode_Daerah         14306
Sudah_Asuransi      14229
Umur_Kendaraan      14275
Kendaraan_Rusak     14188
Premi               14569
Kanal_Penjualan     14299
Lama_Berlangganan   13992
Tertarik                0
dtype: int64
```

```
[ ]  1 df_test.isnull().sum()
```

```
Jenis_Kelamin       0
Umur                0
SIM                 0
Kode_Daerah         0
Sudah_Asuransi      0
Umur_Kendaraan      0
Kendaraan_Rusak     0
Premi               0
Kanal_Penjualan     0
Lama_Berlangganan   0
Tertarik            0
dtype: int64
```

Pada gambar diatas, dapat diketahui bahwa 10 atribut di dalam dataset kendaraan_train memiliki nilai kosong dan 2 atributnya tidak memiliki nilai kosong sedangkan pada kolom di dataset kendaraan_test tidak memiliki nilai kosong pada semua atribut.

# 4. Data Preprocessing Train & Test

## 4.1 Mengisi nilai kosong

```
# Fill Missing Value
df_train["Jenis_Kelamin"] = df_train["Jenis_Kelamin"].replace(np.NaN, np.random.choice(a=["Pria","Wanita"], p=[0.54,0.46]))
df_train["Umur"] = df_train["Umur"].replace(np.NaN, df_train['Umur'].mean())
df_train["SIM"] = df_train["SIM"].replace(np.NaN, np.random.choice(a=[1, 0],p=[0.9956, 0.0044]))
df_train["Kode_Daerah"] = df_train["Kode_Daerah"].replace(np.NaN, statistics.mode(df_train["Kode_Daerah"]))

df_train["Sudah_Asuransi"] = df_train["Sudah_Asuransi"].replace(np.NaN, np.random.choice(a=[0, 1],p=[0.5417, 0.4583]))
df_train["Umur_Kendaraan"] = df_train["Umur_Kendaraan"].replace(np.NaN, np.random.choice(a=["< 1 Tahun", "1-2 Tahun", "> 2 thaun"
df_train["Kendaraan_Rusak"] = df_train["Kendaraan_Rusak"].replace(np.NaN, np.random.choice(a=["Pernah", "Tidak"],p=[0.5355,0.4645

df_train["Premi"] = df_train["Premi"].replace(np.NaN, df_train["Premi"].mean())
df_train["Kanal_Penjualan"] = df_train["Kanal_Penjualan"].replace(np.NaN, 152)
df_train["Lama_Berlangganan"] = df_train["Lama_Berlangganan"].replace(np.NaN, df_train["Lama_Berlangganan"].mean())

print(df_train.isnull().sum())
```

Pada dataset kendaraan_train terdapat 10 atribut yang memiliki nilai kosong, hal ini akan berdampak atau mempengaruhi dari hasil klasifikasi yang dilakukan. Untuk mengatasinya kami melakukan pengisian terhadap atribut yang memiliki nilai kosong. Pada eksplorasi data sebelumnya untuk dataset kendaraan_train memiliki 10 atribut yang terdapat missing value. Maka kami mengisi atribut-atribut tersebut dengan beberapa pendekatan diantaranya adalah mengatasi missing value dengan nilai mean, nilai random, dan nilai statistik dari nilai atribut tersebut. Maka setelah semua missing value pada kendaraan_train sudah tergantikan dengan beberapa metode mengisi nilai kosong diatas, hasilnya dapat dilihat pada gambar dibawah ini.

```
id                    0
Jenis_Kelamin         0
Umur                  0
SIM                   0
Kode_Daerah           0
Sudah_Asuransi        0
Umur_Kendaraan        0
Kendaraan_Rusak       0
Premi                 0
Kanal_Penjualan       0
Lama_Berlangganan     0
Tertarik              0
dtype: int64
```

## 4.2 Mengubah tipe data

```
ubah_tipe_data = {'Jenis_Kelamin': object,  'Umur': int,  'SIM': int, 'Kode_Daerah':int, 'Sudah_Asuransi': int, 'Umur_Kendaraan':
                  'Kendaraan_Rusak': object,  'Premi':int,  'Kanal_Penjualan':int,  'Lama_Berlangganan':int}
df_train = df_train.astype(ubah_tipe_data)
df_test = df_test.astype(ubah_tipe_data)
df_test.dtypes
```

Sebelum data diproses pada model untuk pengklasifikasian maka nilai yang sebelumnya bertipe float maka diubah menjadi integer. Konversi ini diterapkan pada semua atribut yang bertipe data float.

```
id                    int64
Jenis_Kelamin        object
Umur                float64
SIM                 float64
Kode_Daerah         float64
Sudah_Asuransi      float64
Umur_Kendaraan       object
Kendaraan_Rusak      object
Premi               float64
Kanal_Penjualan     float64
Lama_Berlangganan   float64
Tertarik              int64
dtype: object
```

Sebelum dikonversi

```
Jenis_Kelamin        object
Umur                  int64
SIM                   int64
Kode_Daerah           int64
Sudah_Asuransi        int64
Umur_Kendaraan       object
Kendaraan_Rusak      object
Premi                 int64
Kanal_Penjualan       int64
Lama_Berlangganan     int64
Tertarik              int64
dtype: object
```

Sesudah dikonversi

### 4.3 Pelabelan data kategorial dan concat & drop atribut

```
# Labeling pada data Katagorial - Kendaraan rusak
replaced_value_krusak = {"Tidak":0,"Pernah":1}
df_train["Kendaraan_Rusak"]= df_train["Kendaraan_Rusak"].replace(replaced_value_krusak)
df_test["Kendaraan_Rusak"]= df_test["Kendaraan_Rusak"].replace(replaced_value_krusak)
# Labeling pada data Katagorial - jenis kelamin rusak
replaced_value_jkelamin = {"Wanita":0, "Pria":1}
df_train["Jenis_Kelamin"]= df_train["Jenis_Kelamin"].replace(replaced_value_jkelamin)
df_test["Jenis_Kelamin"]= df_test["Jenis_Kelamin"].replace(replaced_value_jkelamin)
#concat & drop - train
umur_kendaraan = pd.get_dummies(df_train["Umur_Kendaraan"], drop_first=True)
df_train = pd.concat([df_train, umur_kendaraan], axis = 1)
df_train.drop(["Umur_Kendaraan","id","Kode_Daerah","Kanal_Penjualan"], axis=1, inplace=True)
#concat & drop - test
umur_kendaraan = pd.get_dummies(df_test["Umur_Kendaraan"], drop_first=True)
df_test = pd.concat([df_test, umur_kendaraan], axis = 1)
df_test.drop(["Umur_Kendaraan","Kode_Daerah","Kanal_Penjualan"], axis=1, inplace=True)

df_train.head()
```

Melakukan pelabelan data kategorial pada atribut *Jenis_Kelamin* dan Kendaraan_Rusak lalu melakukan concat dan drop data pada kolom *Umur_Kendaraan* dimana nilai(non-numeric) pada kolom tersebut dibuat menjadi sebuah atribut. Jadi pada kolom *Umur_Kendaraan* dilakukan concat menjadi 2 atribut yaitu <1 Tahun dan >2 Tahun.

| | Jenis_Kelamin | Umur | SIM | Sudah_Asuransi | Kendaraan_Rusak | Premi | Lama_Berlangganan | Tertarik | < 1 Tahun | > 2 Tahun |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 30 | 1 | 1 | 0 | 28029 | 97 | 0 | 1 | 0 |
| 1 | 1 | 48 | 1 | 0 | 1 | 25800 | 158 | 0 | 0 | 1 |
| 2 | 0 | 21 | 1 | 1 | 0 | 32733 | 119 | 0 | 1 | 0 |
| 3 | 0 | 58 | 1 | 0 | 0 | 2630 | 63 | 0 | 0 | 0 |
| 4 | 1 | 50 | 1 | 0 | 0 | 34857 | 194 | 0 | 0 | 1 |

# 5. Pemodelan

## 5.1 Logistic Regression

Dalam proses pengklasifikasian terhadap dataset kami menggunakan 2 model dari supervised learning. Model yang pertama yaitu Logistic Regression, Logistic Regression merupakan suatu metode atau model atau algoritma analisis prediktif. Regresi logistik digunakan untuk menggambarkan data dan untuk menjelaskan hubungan antara satu variabel biner (dependen) dan satu atau lebih variabel (independen) nominal, ordinal, interval atau rasio. Model Logistic Regression kali ini dibangun secara scratch dengan beberapa fungsi di dalamnya (class logisticRegression).

- Pertama kami membuat fungsi *__init__* sebagai konstruktor class logisticRegression dengan parameter self untuk beberapa objek didalamnya, lr=0,001 (learning rate nya), dan n_iters = 1000 sebagai perulang pada gradient descent. Didalam fungsi constructor __init__ ada beberapa parameter objek lr, n_iters, weights, dan bias, seperti code dibawah ini :

```
1 class logisticRegression:
2
3   def __init__(self,lr=0.001,n_iters=1000):
4     self.lr = lr
5     self.n_iters = n_iters
6     self.weights = None
7     self.bias = None
```

- Kedua membangun fungsi normalisasi untuk menormalkan input. di dalam fungsi ini terdapat n_sample sebagai jumlah contoh training dan n_features sebagai jumlah fiturnya. Dan karena fungsi ini normalisasi data maka nilai dari n_features kami jadikan atau memformatkannya menjadi 0,.... dengan np.zeros(), seperti code dibawah ini

```
 9   def fit(self,X,y):
10     n_samples, n_features = X.shape
11     self.weights = np.zeros(n_features)
12     self.bias = 0
```

Setelah proses tersebut dilakukan maka saatnya mencari nilai optimal dari parameter lr (learning rate). Dalam perulangan ini terdapat beberapa variabel sebagai penampung nilai dari hasil operasi seperti dw, db.

$$dw = (1/n) * baris\ x\ kolom(XT, (y_{pred} - y))$$

$$db = (1/n) * jumlah\ matriks(y_{pred} - y)$$

dw disini dapat diartikan sebagai turunan parsial yang sehubungan dengan w atau weights. db disini dapat diartikan sebagai turunan parsial yang sehubungan dengan b atau bias. Kemudian untuk mencari nilai optimalnya dapat dihitung dengan: $weight = lr * dw$ dan $bias = lr * db$, maka dapat diimplementasikan seperti kode dibawah ini :

```
    for _ in range(self.n_iters):
      linear_model = np.dot(X,self.weights) + self.bias
      y_predicted = self._sigmoid(linear_model)
      dw = (1/n_samples) * np.dot(X.T,(y_predicted-y))
      db = (1/n_samples) * np.sum(y_predicted-y)
      self.weights -= self.lr *dw
      self.bias -= self.lr * db
```

- Selanjutnya yaitu membangung fungsi Prediksi atau def predict. Pertama yang dihitung adalah precision yaitu baris kali kolom X dengan weights lalu ditambang dengan bias. Kemudian menghitung nilai prediksi nya dengan ketentuan apabila y >= 0.5 maka pembulatan ke atas 1 atau jika < 0.5 maka pembulatan ke 1 (di dalam perulangan semasa nilai y_predicted), Implementasi code seperti dibawh ini:

```
def predict(self,X):
  linear_model = np.dot(X,self.weights) + self.bias
  y_predicted = self._sigmoid(linear_model)
  y_predicted_cls = [1 if i>0.5 else 0 for i in y_predicted]
  return y_predicted_cls
```

- Selanjutnya membuat fungsi sigmoid atau def _sigmoid. Fungsi Sigmoid akan menekan semua inputnya (nilai pada sumbu x) antara 0 dan 1 pada parameter x. Dengan formula dan implementasi code seperti dibawah ini:

$$g(z) = \frac{1}{1 + e^{-z}}$$

```
def _sigmoid(self,x):
  return(1/(1+np.exp(-x)))
```

## 5.2 Linear Regression

Dalam proses pengklasifikasian terhadap dataset kami selanjutnya menggunakan model dari supervised learning yaitu Linear Regression, Linear Regression adalah suatu metode atau model atau algoritma prediktif hampir sama dengan pada Model Logistic Regression. Linear Regression juga dapat diartikan metode untuk memodelkan hubungan antara dua variabel dengan menyesuaikan persamaan linier dengan data yang sedang diamati. Satu variabel dianggap sebagai variabel penjelas, dan yang lainnya dianggap sebagai variabel terikat. Pada model Linear Regression kali ini dibangun secara scratch dengan beberapa fungsi di dalamnya (class linearRegression).

- Sama seperti pertama dilakukan saat membangun model Logistic Regression kami membuat fungsi *__init__* sebagai konstruktor class linearRegression dengan parameter self untuk beberapa objek didalamnya, learning_rate=0,001 (learning rate nya), dan n_iters = 1000 sebagai perulangan pada gradient descent. Didalam fungsi constructor __init__ ada beberapa parameter objek lr, n_iters, weights, dan bias, seperti code dibawah ini :

```
class linearRegression:

    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None
```

- Kedua membangun fungsi normalisasi untuk menormalkan input. di dalam fungsi ini terdapat n_sample sebagai jumlah contoh training dan n_features sebagai jumlah fiturnya. Dan karena fungsi ini normalisasi data maka nilai dari n_features kami jadikan atau memformatkannya menjadi 0,.... dengan np.zeros(), seperti code dibawah ini

```
 9     def fit(self, X, y):
10         n_samples, n_features = X.shape
11         self.weights = np.zeros(n_features)
12         self.bias = 0
```

Masih sama dengan proses Logistic Regression yaitu setelah proses diatas tersebut dilakukan maka saatnya mencari nilai optimal dari parameter lr (learning rate). Dalam perulangan ini terdapat beberapa variabel sebagai penampung nilai dari hasil operasi seperti dw, db.

$$dw = (1/n) * baris\ x\ kolom(XT, (y_{pred} - y))$$
$$db = (1/n) * jumlah\ matriks(y_{pred} - y)$$

dw disini juga dapat diartikan sebagai turunan parsial yang sehubungan dengan w atau weights. db disini dapat diartikan sebagai turunan parsial yang sehubungan dengan b atau bias. Kemudian untuk mencari nilai optimalnya dapat dihitung dengan: $weight\ -= lr * dw$ dan $bias\ -= lr * db$, maka dapat diimplementasikan seperti kode dibawah ini :

```
14         # gradient descent
15         for _ in range(self.n_iters):
16             # Menghitung prediksi: y_pred atau h(x)
17             y_predicted = np.dot(X, self.weights) + self.bias
18             # compute gradients
19             # Menghitung turunan dari parameter (bobot, dan bias)
20             dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y))
21             db = (1 / n_samples) * np.sum(y_predicted - y)
22 
23             # update parameters
24             self.weights -= self.lr * dw
25             self.bias -= self.lr * db
```

- Selanjutnya yaitu membangung fungsi Prediksi atau def predict di dalam class linearRegression, yaitu mengembalikan nilai dari penjumlahan baris x kolom(X dan weights ) + bias, maka dapat terimplementasikan seperti code dibawah ini:

```
28     def predict(self, X):
29         y_approximated = np.dot(X, self.weights) + self.bias
30         return y_approximated
```

- Kemudian kita tambahkan fungsi untuk menghitung MSE (mean squared error) dengan implementasi formulasi dan code seperti dibawah ini:ŷ

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

```
32 def mean_squared_error(y_true, y_pred):
33     return np.mean((y_true - y_pred)**2)
```

# 6. Evaluasi Akurasi Model

## 6.1 Akurasi Logistic Regression

Kami menguji model Logistic Regression yang telah dibangun dengan menghitung beberapa nilai *seperti precision, recall, f1-score, akurasi* nya menggunakan method *classification_report* dari library *sklearn.metrics*. Kemudian kami menghitung akurasi pada model terhadap data set *kendataraan_train* dan *kendaraan_test*, dengan implementasi dan output sebagai berikut:

```
#Feature Selection
X1 = df_train.iloc[:,[0,1,2,3,4,5,6,7,8,9]].values
X2 = df_test.iloc[:,[0,1,2,3,4,5,6,7,8,9]].values

y1 = df_train.iloc[:,7].values
y2 = df_test.iloc[:,7].values

#Scale data
min_max_scaler = preprocessing.MinMaxScaler()
X1 = min_max_scaler.fit_transform(X1)
X2 = min_max_scaler.fit_transform(X2)

#Split data
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.3)

log_model = logisticRegression()
log_model.fit(X_train, y_train)

log_predict1 = log_model.predict(X_test)
log_predict2 = log_model.predict(X2)

classes2 = ['Tidak Tertarik', 'Tertarik']
print(classification_report(y_test, log_predict1, target_names=classes2)); var_handle = y_test
print("Akurasi kendaraan_train dari model :" , accuracy_score(y_test,log_predict1),"\n")

print(classification_report(y2, log_predict2, target_names=classes2))
print("Akurasi kendaraan_test dari model :" , accuracy_score(y2,log_predict2),"\n")
```

```
                precision    recall  f1-score   support

Tidak Tertarik       0.88      1.00      0.93     75209
      Tertarik       0.00      0.00      0.00     10541

      accuracy                           0.88     85750
     macro avg       0.44      0.50      0.47     85750
  weighted avg       0.77      0.88      0.82     85750

Akurasi kendaraan_train dari model : 0.8770728862973761

                precision    recall  f1-score   support

Tidak Tertarik       0.88      1.00      0.93     41778
      Tertarik       0.00      0.00      0.00      5861

      accuracy                           0.88     47639
     macro avg       0.44      0.50      0.47     47639
  weighted avg       0.77      0.88      0.82     47639

Akurasi kendaraan_test dari model : 0.8769705493398267
```

Dari hasil output pada gambar diatas, model memiliki nilai akurasi sebesar 0,88. Kemudian akurasi model terhadap dataset kendaraan_train sebesar 0.8770728862973761 dan akurasi model terhadap dataset kendaraan_test sebesar 0.8769705493398267. Serta pada training kendaraan_train orang yang Tertarik ada 10449 orang dan yang tidak tertarik ada 75301. Kemudian pada testing kendaraan_test, orang yang tidak tertarik ada 41778 dan orang yang tertarik ada 5861.

### 6.2 Akurasi Linear Regression

Kemudian kami menguji model Linear Regression yang telah dibangun dengan tidak menghitung beberapa nilai *seperti precision, recall, f1-score, akurasi* nya menggunakan yang method *classification_report* dari library *sklearn.metrics* seperti halnya pada Logistic Regression. Kami mengukur mean squared error dari model Linear Regression dan untuk menghitung akurasi pada model Linear Regression dengan menghitung score variansinya sebagai titik ukur akurasi modelnya, Perhitungan menggunakan *explained_variance_score()* dari sklearn metrics terhadap dataset *kendaraan_train* dan *kendaraan_test*. Untuk implementasinya dan outputnya seperti dibawah ini.

```
#Feature Selection
X1 = df_train.iloc[:,[0,1,2,3,4,5,6,7,8,9]].values
X2 = df_test.iloc[:,[0,1,2,3,4,5,6,7,8,9]].values

y1 = df_train.iloc[:,7].values
y2 = df_test.iloc[:,7].values
#Scale data
min_max_scaler = preprocessing.MinMaxScaler()
X1 = min_max_scaler.fit_transform(X1)
X2 = min_max_scaler.fit_transform(X2)

#Split data
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.2, random_state=1234)

lin_model = linearRegression(learning_rate=0.01, n_iters=1000)
lin_model.fit(X_train, y_train)

lin_predict1 = lin_model.predict(X_test)
lin_predict2 = lin_model.predict(X2)

mse = mean_squared_error(y_test, lin_predict1)
print("MSE:", mse)

from sklearn import metrics
train = metrics.explained_variance_score(y_test, lin_predict1)
test = metrics.explained_variance_score(y2, lin_predict2)
print("Akurasi kendaraan_train dari model :" , train,"\n")
print("Akurasi kendaraan_test dari model :" , test,"\n")
```

```
MSE: 0.014043692761516 64
Akurasi kendaraan_train dari model : 0.8687035404404303

Akurasi kendaraan_test dari model : 0.8694707016814271
```

Dari hasil output pada gambar diatas, model memiliki nilai MSE sebesar 0.014043692761516 64. Kemudian akurasi model terhadap dataset *kendaraan_train* sebesar 0.8687035404404303 dan akurasi model terhadap dataset *kendaraan_test* sebesar 0.8694707016814271.

## 7. Eksperimen

### 7.1 Perbandingan Akurasi Library & Model Logistic Regression

Berikut merupakan perbandingan akurasi library dan akurasi model pada logistic regression. Dengan library, didapatkan akurasi pada kendaraan_train yaitu 1.0 dan pada kendaraan_test yaitu 1.0. Sedangkan dengan model, didapatkan akurasi pada kendaraan_train yaitu 0.8773877551020408 dan pada kendaraan_test yaitu 0.8769705493398267.

```
Akurasi lib Logistic Regression

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     75236
           1       1.00      1.00      1.00     10514

    accuracy                           1.00     85750
   macro avg       1.00      1.00      1.00     85750
weighted avg       1.00      1.00      1.00     85750

Akurasi pada kendaraan_train : 1.0
Akurasi pada kendaraan_test : 1.0

------------------------------------------------------------
Akurasi model Logistic Regression

              precision    recall  f1-score   support

           0       0.88      1.00      0.93     75236
           1       0.00      0.00      0.00     10514

    accuracy                           0.88     85750
   macro avg       0.44      0.50      0.47     85750
weighted avg       0.77      0.88      0.82     85750

Akurasi kendaraan_train dari model : 0.8773877551020408
Akurasi kendaraan_test dari model : 0.8769705493398267
```

### 7.2 Perbandingan Akurasi Library & Model Linear Regression

Berikut merupakan perbandingan akurasi library dan akurasi model pada Linear Regression. Dengan library, didapatkan Mean Squared Errornya 2.1788198608901004 dan akurasi pada *kendaraan_train* yaitu 1.0 dan pada *kendaraan_test* yaitu 1.0. Sedangkan dengan model, didapatkan Mean Squared Errornya 0.01404369276151664 akurasi pada *kendaraan_train* yaitu 0.8687035404404303 dan *pada kendaraan_test* yaitu 0.8694707016814271.

```
Akurasi lib Linear Regression
MSE: 2.1788198608901004e-31
Akurasi kendaraan_train dari model : 1.0
Akurasi kendaraan_test dari model : 1.0

Akurasi model Linear Regression
MSE: 0.01404369276151664
Akurasi kendaraan_train dari model : 0.8687035404404303
Akurasi kendaraan_test dari model : 0.8694707016814271
```

## Kesimpulan

| Dataset | Akurasi model yang dibangun | |
|---|---|---|
| | Logistic Regression | Linear Regression |
| *kendaraan_train* | 0.8777492711370263 | 0.8687035404404303 |
| *kendaraan_test* | 0.8769705493398267 | 0.8694707016814271 |

Dari data diatas dapat disimpulkan dengan melihat tingkat akurasi di kedua model hampir sama antara model Logistic Regression dengan Linear Regression. Tetapi pada model Logistic Regression memiliki akurasi di atas akurasi model Linear Regression terhadap dataset train & test. Yang berarti model terbaik yaitu Logistic Regression karena score akurasinya lebih unggul.

Pada Logistic Regression juga dapat memberikan classification report yang merupakan informasi hasil klasifikasi yang didapat. Pada Logistic Regression untuk klasifikasi Tidak tertarik dan Tertarik dapat dilihat pada tabel berikut:

| Hasil klasifikasi pada model Logistic Regression | | |
|---|---|---|
| Dataset | Tidak Tertarik | Tertarik |
| *kendaraan_train* | 75301 | 10449 |
| *kendaraan_test* | 41778 | 5861 |

## Lampiran

**Laporan, Dataset dan Code Program jupyter notebook :**
Github : https://github.com/fasa2297/fml-ml2-classification
GDrive : https://bit.ly/3E7cQUe

## Referensi


- [Verma, 2021] Verma Suraj (2021). *Logistic Regression, Linear Regression in Python*.
- [Allison, 2014] Allison Paul D (2014). *Measures of Fit for Logistic Regression.*
- https://stackoverflow.com/