Laporan Tugas Besar 2 Intelegensi Artifisial

Implementasi Algoritma Pembelajaran Mesin

Semester I Tahun 2024/2025

Oleh:

Venantius Sean Ardi Nugroho	**13522078**
Muhammad Atpur Rafif	**13522086**
Andhika Tantyo Anugrah	**13522094**
M. Hanief Fatkhan Nasrullah	**13522100**

PROGRAM STUDI INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

Bab 1
Implementasi KNN

Algoritma *Nearest Neighbor* adalah salah satu algoritma pembelajaran *supervised* pada pembelajaran mesin untuk klasifikasi. Sebelum diberikan label klasifikasi, pertama-tama dilakukan perhitungan jarak antara objek dengan data latih. Kemudian diambil sebanyak *k* kemunculan dari data latih yang paling mirip dengan objek. Lalu cari label klasifikasi yang paling banyak jumlah kemunculannya dibandingkan kandidat label lain (*plurality vote*). Terakhir, mengembalikan label klasifikasi untuk objek sebagai hasil prediksi dari KNN.

Pada tugas besar 2 ini, algoritma KNN diimplementasikan dengan membuat kelas KNN. Kelas ini memiliki atribut *k*, *features*, dan *target*. Kelas ini juga memiliki beberapa *method* seperti *fit*, *_calculate_distance*, dan *predict*. Metode-metode publik diimplementasikan mengikuti definisi KNN di atas.

Pelatihan dengan algoritma KNN tidak menghasilkan model, karena KNN bersifat *lazy learner* yang hanya menyimpan semua data latih dan tidak memiliki hipotesis. Dengan mengikuti sifat KNN ini, *method fit* hanya melakukan penyimpanan *features* dan *target* untuk dibandingkan dengan data uji nantinya. *Method _calculate_distance* menghitung jarak dari data latih dengan data uji. *Method predict* menggunakan *priority queue* untuk menyimpan jarak dari data uji dengan data latih. *Priority queue* ini mengurutkan data berdasarkan jarak yang terdekat, lalu diambil *k* data pertama untuk dihitung jumlah kemunculan label yang disimpan pada *dictionary* yang bernama *counter*. Pada akhirnya, dihitung probabilitas untuk setiap label yang ada di *counter* ini dengan cara membagi jumlah kemunculannya dengan *k*.

Bab 2
Implementasi Naive-Bayes

Algoritma Naive-Bayes adalah salah satu algoritma pembelajaran *supervised* pada pembelajaran mesin untuk klasifikasi. Tidak seperti KNN yang menyimpan data latih, Naive-Bayes hanya menyimpan probabilitas yang sudah dihitungnya pada saat *fitting*. Untuk mendapatkan hipotesis atau modem probabilitas, dilakukan pelatihan. Pertama-tama dilakukan pengukuran frekuensi kemunculan setiap nilai atribut untuk label tertentu, kemudian semua frekuensi dari setiap label. Lalu, dihitung probabilitas untuk $P(a_i|v_j)$ dan akhirnya, dihitung probabilitas untuk $P(v_j)$.

Proses klasifikasi dilakukan dengan menghitung prediksi probabilitas atribut untuk setiap label

$$\Pi_i P(a_i|v_j)$$

Kemudian kalikan hasilnya dengan probabilitas setiap label

$$P(v_j|a_1, a_2, \ldots, a_n) = P(v_j) . \Pi_i P(a_i|v_j)$$

Lalu, diambil label dengan probabilitas $P(v_j|a_1, a_2, \ldots, a_n)$ yang terbesar dan dijadikan sebagai prediksi terbaik.

Algoritma KNN diimplementasikan dengan membuat kelas NaiveBayes. Kelas ini memiliki atribut *data_count*, *feature_count*, *target_class*, *target_counter*, *target_probability*, *feature_counter*, dan *feature_probability*. Kelas ini juga memiliki *method fit* dan *predict*.

Pelatihan dengan algoritma Naive-Bayes dilakukan pada *method fit*. *Method* ini pertama-tama melakukan perhitungan frekuensi label yang terdapat pada *target* dan dilakukan perhitungan probabilitas label $P(v_j)$. Kemudian, dilakukan perhitungan frekuensi kemunculan setiap nilai atribut dan dihitung probabilitas untuk kemunculan setiap nilai atribut untuk label tertentu $P(a_i|v_j)$. Prediksi label dengan algoritma ini dilakukan pada *method predict*. *Method* ini menggunakan cara yang sama dengan cara prediksi yang sudah dibahas sebelumnya, yaitu dengan menghitung prediksi probabilitas atribut untuk setiap label, mengalikannya dengan probabilitas setiap label, dan mengambil probabilitas tertinggi sebagai prediksi label untuk data uji tersebut.

Bab 3
Implementasi ID3

Decision Tree Learning (DTL) adalah salah satu pendekatan dalam pembelajaran mesin *supervised* pada pembelajaran mesin untuk klasifikasi. Pohon keputusan digunakan sebagai model untuk mengambil keputusan mengenai observasi pada data latih. Pohon keputusan dibentuk dari *node* keputusan dan daun keputusan. Sebuah *node* dapat memiliki 2 atau lebih cabang yang merepresentasikan nilai dari atribut yang diuji. Daun keputusan menghasilkan hasil yang homogen.

Iterative Dichotomiser 3 (ID3) adalah salah satu algoritma dengan pendekatan *greedy*. Dalam membentuk sebuah pohon keputusan atau training, selalu dicari information gain tertinggi untuk setiap atribut yang ada. Information gain tersebut dihitung dengan menggunakan entropy.

```
def entropy(self,data):
      value,counts = np.unique(data, return_counts=True)
      proportion = counts/len(data)
      return -np.sum([p*np.log2(p) for p in proportion if p>0])
```

Entropy dihitung dengan mencari frekuensi kemunculan setiap nilai yang unik pada data. Data yang dimasukkan adalah data target (“attack_cat”). Kemudian, hitung proporsi dari setiap nilai yang muncul, dikalikan dengan log proporsi. Terakhir, jumlahkan semua nilai tersebut dan dikalikan negatif satu.

Entropy pertama yang dihitung adalah data awal yang belum diubah selain dari proses cleaning and preprocessing. Entropy tersebut kemudian akan dikurangkan dengan proporsi dikali entropy untuk setiap nilai pada atribut.

```
for value in featureValues:
                  splittedX,splittedY = self.splitData(X,y,feature,value)
                  valueEntropy = self.entropy(splittedY)
                  proportion = len(splittedY)/len(y)
                  featureEntropy += proportion*valueEntropy
```

Bagian yang ditandai dengan splitData merupakan bagian untuk melakukan pemecahan dan mencari seluruh baris pada X dan Y yang setiap atributnya memiliki nilai sama dengan value pada parameter tersebut. Pada parameter tersebut, value merupakan seluruh nilai unik yang ada pada atribut.

```
def splitData(self,X,y,feature,value):
      filteredIndex = np.where(X[:, feature] == value)[0]
      return X[filteredIndex],y[filteredIndex]
```

Setelah dilakukan pemecahan, kemudian dihitung nilai gain dengan mengurangi entropy target dengan proporsi dikali entropy setiap value pada atribut terpilih dan dicari yang paling tinggi. Jika nilai gain tertinggi lebih dari nol, maka atribut tersebut dipilih menjadi atribut penentu dan ditambahkan child node dengan setiap child node adalah seluruh nilai unik dari atribut terpilih. Untuk setiap child node, lakukan ulang operasi yang sama. Pada dasarnya operasi ini dilakukan secara rekursif sampai base case terpenuhi, yaitu seluruh target memiliki nilai yang sama.

```
if len(set(y)) == 1:
            self.result = y[0]
            return
```

Jika ketika melakukan pemilihan gain dan seluruhnya memiliki nilai nol, maka akan dipilih atribut yang belum pernah terpilih sebelumnya secara random. Pemilihan ini harus memastikan atribut yang terpilih belum pernah terpilih sebelumnya. Jika sudah tidak ada atribut yang dapat dipilih, maka hasil yang diberikan adalah nilai mayoritas yang ada pada target.

```
if maxGain > 0:
            self.feature = maxCriteria
            self.chosen.add(maxCriteria)
            featureValues = set(X[:,maxCriteria])
            for values in featureValues:
                  newInstance = ID3(child={},chosen=self.chosen)
                  splittedX,splittedY =
self.splitData(X,y,self.feature,values)
                  self.child[values] = newInstance
                  self.child[values].fit(splittedX,splittedY)
      else:
            RNG = [i for i in range(len(X[0])) if i not in self.chosen]
            if len(RNG)==0:
                  self.result = self.majority
            else:
                  chosenRNG = choice(RNG)
                  self.chosen.add(chosenRNG)
                  self.feature = chosenRNG
                  maxCriteria = chosenRNG
                  featureValues = set(X[:,maxCriteria])
```

```
                for values in featureValues:
                newInstance = ID3(child={},chosen=self.chosen)
                splittedX,splittedY =
self.splitData(X,y,self.feature,values)
                self.child[values] = newInstance
                self.child[values].fit(splittedX,splittedY)
```

Untuk melakukan prediksi, pencarian dilakukan secara rekursif juga

```
def predictRow(self,row):
      if self.result:
            return self.result
      else:
            valueChosen = row[self.feature]
            if valueChosen in self.child:
                  return self.child[valueChosen].predictRow(row)
            else:
                  return self.majority
```

Untuk satu baris, diperiksa terlebih dahulu pada pohon, manakah atribut yang terbaik. Nilai atribut dari baris tersebut diperiksa apakah ada dalam decision tree. Jika ada, periksa lagi childnya dengan operasi yang sama secara rekursif sampai base case ditemukan, yaitu node memiliki suatu nilai target. Jika terdapat kasus nilai atribut yang diambil tidak ada pada decision tree, maka akan dikembalikan nilai yang paling sering muncul pada target.

Bab 4
Cleaning dan *Preprocessing*

Data yang digunakan untuk *training* dan yang ingin diklasifikasi perlu diolah terlebih dahulu agar algoritma pembelajaran mesin yang digunakan bisa mengolahnya serta menghindari *overfitting* dan *underfitting*. Selain kedua alasan tersebut *data cleaning* dan *preprocessing* juga digunakan untuk mengurangi waktu komputasi yang diperlukan. Untuk itu, dibuat sebuah *pipeline* yang memudahkan organisasi kelas - kelas yang digunakan dalam transformasi data menjadi yang sesuai. Proses *cleaning* dan *preprocessing* yang dilakukan terdiri:

- Handling Missing Data
  Dalam penanganan pada data yang hilang, metode yang digunakan oleh penulis adalah imputasi dengan mean pada fitur - fitur numerical, sedangkan imputasi dengan modus pada fitur - fitur categorical. Metode tersebut memiliki beberapa kelebihan, selain sederhana dan cepat dalam implementasi dan waktu komputasi, metode ini unggul dibanding metode berbasis penghapusan karena menjaga kelengkapan data. Metode ini juga memiliki kelebihan dibanding metode berbasis interpolasi dalam hal mengurangi *overfitting*.
- Handling Outliers
  Metode yang digunakan dalam penanganan outlier adalah dengan melakukan flooring ke kuartil pertama dan capping pada kuartil ketiga bila suatu data melewati threshold tersebut. Alasan penggunaan metode ini adalah karena dengan metode ini, kelengkapan dan distribusi data dapat terjaga.
- Handling Duplicates
  Data - data duplikat yang terdapat pada dataset, di-drop dari dataset. Hal tersebut digunakan terutama untuk mengurangi *overfitting* serta mengurangi sumber daya komputasi yang diperlukan untuk menggunakan model.
- Feature Encoding
  Metode yang digunakan adalah *one hot encoding*, metode ini cukup sederhana, untuk tiap kategori pada fitur categorical, buat satu kolom yang merepresentasikan apakah untuk kategori tersebut, barisnya termasuk kategori tersebut atau bukan. Untuk mengerti alasan penggunaan *one hot encoding*, perlu diketahui bahwa encoding pada categorical feature pada tugas ini terutama digunakan untuk mencari korelasi antara suatu fitur dengan target. One hot encoding cocok untuk hal tersebut karena sangat minim asumsi yang digunakan, dibanding mungkin target encoding. Kalkulasi korelasi diharapkan dapat lebih akurat dengan adanya sifat tersebut.
- Feature Engineering
  Feature engineering yang dilakukan oleh penulis adalah feature selection. Pada dasarnya metode yang digunakan untuk feature yang relevan adalah sebagai berikut :
    - untuk tiap feature pada numerical feature, cari rata - rata korelasinya terhadap target.

    - untuk tiap feature pada categorical feature yang sudah di encode cari rata korelasinya terhadap target, lalu cari nilai maksimal dari rata - rata tersebut sebagai nilai korelasi suatu fitur categorical terhadap target
    - pilih feature yang korelasi terhadap targetnya melebihi threshold tertentu.

  Alasan penggunaan metode ini untuk menghilangkan fitur yang tidak memiliki korelasi tinggi dengan target
- Feature Scaling

  Feature scaling termasuk tahap yang penting dalam *data preprocessing*. Hal tersebut berguna untuk menjaga keseragaman kontribusi pada tiap fitur. Pada tugas ini digunakan metode Min Max Scaling karena menjaga korelasi antar fitur dan cukup sederhana.
- Handling Imbalanced Dataset

  Metode yang digunakan untuk menangani data set yang tidak seimbang adalah dengan Menggunakan gabungan antara oversampling dan undersampling karena merupakan jalan tengah dari kedua solusi. Terdapat sebuah kelas target yang memiliki titik data yang sangat kecil. Penggunaan undersampling akan mengakibatkan data menjadi 0.6% dari data awal, hal ini akan mengurangi akurasi. Sedangkan oversampling akan membuat data menjadi tiga kali lipat lebih besar dan akan menggunakan komputasi yang besar.
- Dimensionality Reduction

  Data ditransformasikan menjadi kategorikal karena kesamaan dari tiga algoritma adalah penggunaan data kategorikal. Hal ini terutama untuk Naive Bayes dan ID3. Dimensi yang berupa bilangan numerikal, atau bilangan real dijadikan kategori menggunakan cara binning. Metode ini dijalankan dengan mendefinisikan banyaknya bin yang dibuat. Setiap bin akan menjadi sebuah kategori baru.

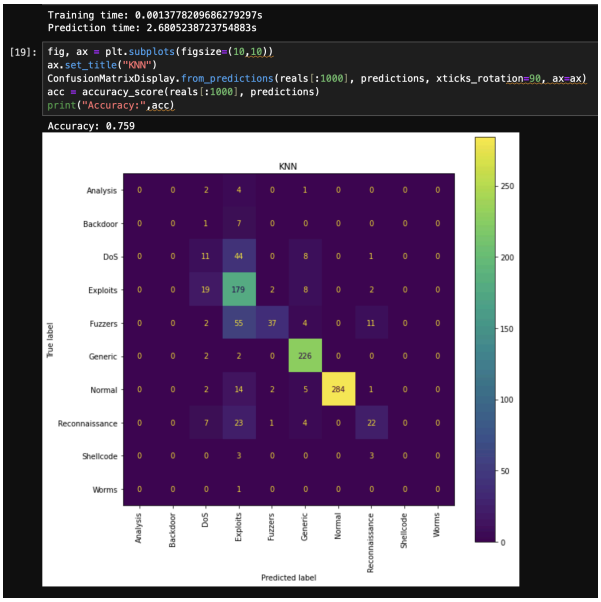
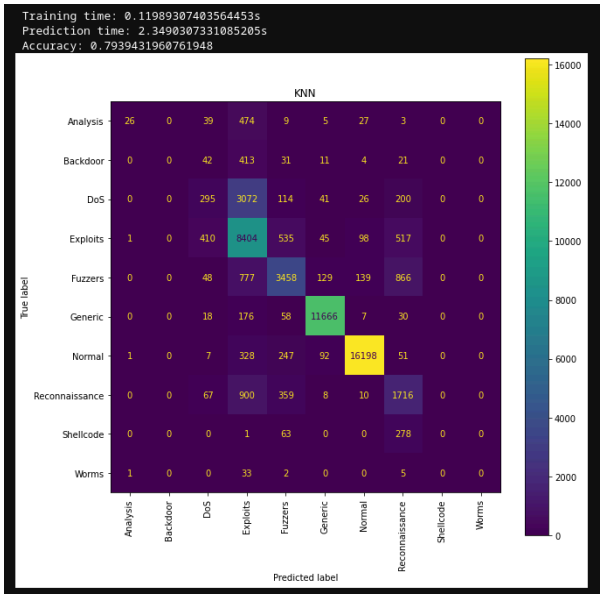
Bab 5
Eksperimen

Global

Terdapat sebuah tahap pada pipeline yang menghasilkan akurasi yang rendah jika digunakan. Tahapan tersebut adalah sampler. Metode yang kami gunakan adalah pertengahan antara undersampling dan oversampling. Bentuk data yang dihasilkan setelah menggunakan sampel, hal tersebut tidak merepresentasikan kejadian serangan dengan baik. Informasi mengenai frekuensi terjadinya serangan berpengaruh terhadap tingkat akurasi model. Secara umum, hasil dari library memiliki akurasi, waktu training, dan waktu prediksi yang lebih baik dibandingkan dengan hasil dari yang dibuat secara manual. Namun, margin dari ketiga *metrics* tersebut juga secara umum memiliki margin yang tidak jauh berbeda. Khusus untuk perhitungan dengan Native Bayes, hasil yang dibuat secara manual oleh kelompok kami memiliki akurasi yang sedikit lebih baik dibandingkan dengan hasil dari library.

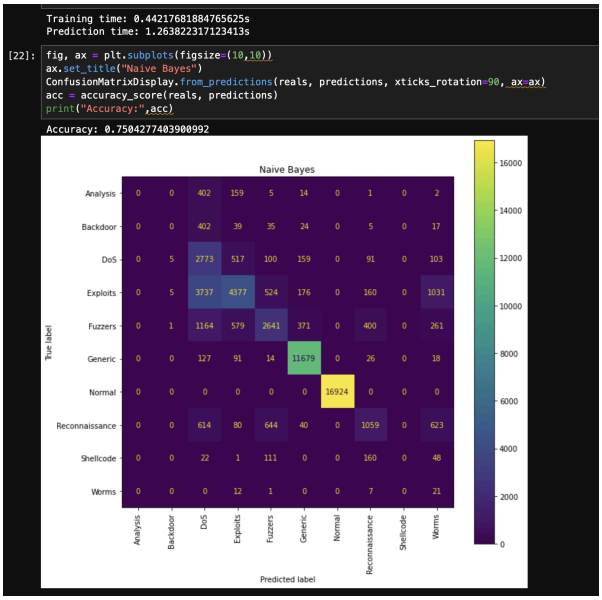
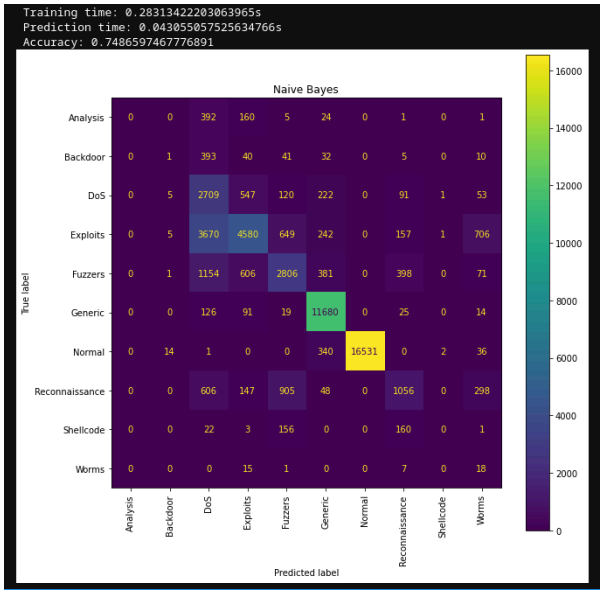
KNN

Perbedaan yang paling terlihat saat mencoba membandingkan algoritma pustaka dengan algoritma yang dibuat secara manual adalah waktu pelatihan yang jauh lebih cepat dari algoritma yang dibuat secara manual. Nilai dari akurasi juga relatif tidak jauh antara penggunaan library dengan manual. Namun terdapat sebuah permasalahan, yaitu waktu yang dibutuhkan untuk algoritma manual sangatlah banyak. Sehingga hanya dapat diperiksa sebanyak 1000 data pertama. Selain itu, tidak terdapat training time, namun komputasi banyak dilakukan pada bagian prediction.

Hasil Manual	Hasil Library
	

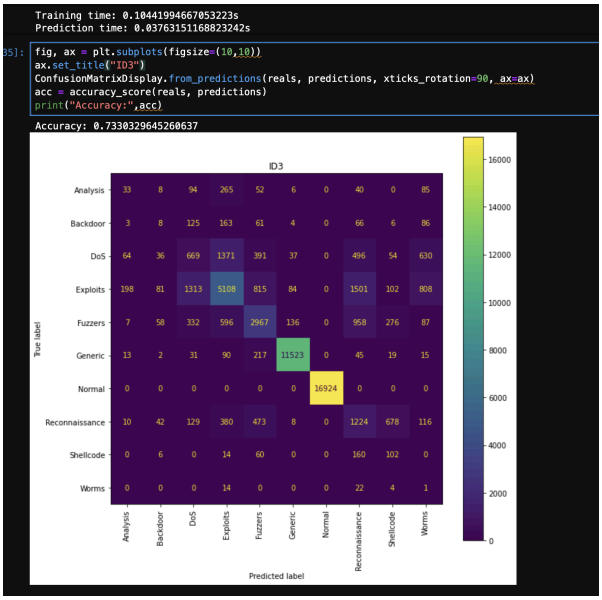
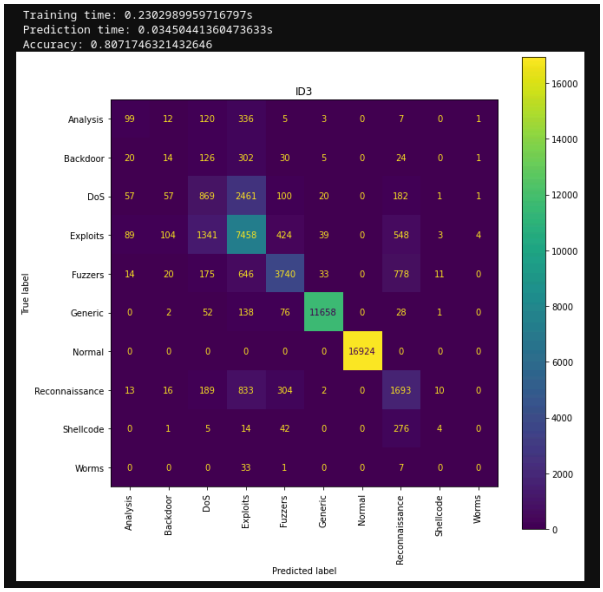
Naive Bayes

Setelah melakukan beberapa permutasi percobaan pipeline. Saat seluruh pipeline (kecuali sampler) dinyalakan, akurasi yang dihasilkan sekitar 40%. Namun saat tahapan duplicate dihapus, tingkat akurasi naik menjadi 70%. Hal ini dikarenakan frekuensi terjadinya serangan merupakan hal yang mempengaruhi hasil model menggunakan algoritma ini. Sama seperti algoritma lainnya, nilai dari penggunaan library sama dengan pembuatan algoritma secara manual.

Hasil Manual	Hasil Library
	

DTL

Sama seperti Naive Bayes, saat *deduplication* dimatikan, tingkat akurasi meningkat, dari 67% ke 73%. Hal ini dikarenakan frekuensi terjadinya serangan mempengaruhi kualitas model yang dihasilkan menggunakan algoritma ini, walaupun dampaknya tidak separah Naive Bayes. Akurasi yang dikeluarkan oleh hasil manual memiliki hasil yang lebih rendah dibandingkan dengan library. Untuk waktu prediksi hasilnya cukup mirip dan waktu training manual memiliki waktu dua kali lebih cepat dibandingkan dengan hasil library.

Hasil Manual	Hasil Library
	

Bab 6
Kontributor

NIM	Nama	Kontribusi
13522078	Venantius Sean Ardi Nugroho	*Cleaning* dan *Preprocessing*, laporan *cleaning* dan *preprocessing*
13522086	Muhammad Atpur Rafif	*Preprocessing* dan Naive Bayes, perbaikan KNN
13522094	Andhika Tantyo Anugrah	KNN, laporan KNN, laporan Naive Bayes, dan sedikit laporan DTL
13522100	M. Hanief Fatkhan Nasrullah	DTL, laporan DTL

Bab 7
Referensi

1. https://sebastianraschka.com/pdf/lecture-notes/stat479fs18/02_knn_notes.pdf
2. https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250331293_IF3170_Materi09_Seg01_AI-kNN.pdf
3. https://www.geeksforgeeks.org/k-nearest-neighbours/
4. https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/210071-Supervised-Learning/90133-Supervised-Learning/1699250380758_IF3170_Materi09_Seg02_AI-NaiveBayes.pdf
5. https://athena.ecs.csus.edu/~mei/177/ID3_Algorithm.pdf
6. https://cdn-edunex.itb.ac.id/64464-Artificial-Intelligence-Parent-Class/298936-Modeling-Decision-Tree-Learning-DTL/120485-Modul-Introduction-to-DTL/1731401412073_IF3170_SupervisedLearning_DTL.pdf
7. https://www.geeksforgeeks.org/iterative-dichotomiser-3-id3-algorithm-from-scratch/