IF3170 Inteligensi Artifisial

Tugas Besar 2

Implementasi Algoritma Pembelajaran Mesin

Disiapkan oleh:

Kelompok 9 - One Week Challenge

1. Ariel Herfrison 13522002
2. Irfan Sidiq Permana 13522007
3. Akbar Al Fattah 13522036
4. Diero Arga Purnama 13522056

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024/2025

Daftar Isi

Implementasi Algoritma Klasifikasi

1. K-Nearest Neighbors (KNN)

Algoritma K-Nearest Neighbors (KNN) adalah salah satu metode *supervised learning* yang digunakan untuk mengklasifikasi data. Pada implementasi yang dibuat, algoritma KNN melakukan klasifikasi menggunakan jarak antara data masukan dengan data latih yang ada. Parameter utama yang digunakan dalam algoritma ini adalah jumlah tetangga terdekat yang akan dicari (*k*), metode penghitungan jarak (*manhattan*, *euclidean*, atau *minkowski*), serta nilai eksponen (*p*) jika metode penghitungan jarak yang dipilih adalah metode Minkowski Distance.

Karena KNN adalah algoritma klasifikasi yang tidak memiliki sebuah hipotesis, untuk pelatihan model, algoritma sekedar menyimpan data latihan menggunakan metode *fit*. Setelah data latihan disimpan, algoritma akan menghitung jarak antara data latihan dengan data yang ingin diklasifikasi. Jarak antar data dihitung menggunakan metode penghitungan yang telah dimasukkan.

Kemudian, algoritma akan memilih *k* data tetangga yang memiliki jarak terkecil. Kelas data yang paling sering muncul di antara *k* tetangga tersebut akan dikembalikan sebagai hasil prediksi. Jika terdapat lebih dari satu kelas yang memiliki jumlah terbesar, maka kelas yang akan dikembalikan adalah kelas yang pertama kali muncul di set data.

Implementasi ini juga memiliki fitur penyimpanan model ke dalam file JSON menggunakan metode *save* yang akan menyimpan parameter-parameter yang telah dimasukkan beserta training data yang digunakan. Model yang telah disimpan dapat dimuat kembali menggunakan metode *load*.

Berikut merupakan *source code* kelas KNN:

knn.py

```
from typing import List, Dict, Optional, Any, Union
import pandas as pd
import numpy as np
import json

class KNN:
    def __init__(self, k: int, method: str = "manhattan",
                 p: Optional[int]  = None) -> None:
        self.k: int                     = k
        self.method: str                = method
        self.p: Optional[int]           = p

        self.train_X: Optional[pd.DataFrame] = None
        self.train_y: Optional[pd.Series]    = None

    @staticmethod
    def getNumericalFeatures(df: pd.DataFrame) -> pd.Index:
        return df.select_dtypes(include = ["float64", "int64", "int32"]).columns
```

```
    @staticmethod
    def calcMinkowskiDist(train_X: pd.DataFrame, inp: pd.Series, p: int = 1) ->
pd.Series:
        numerical_features: pd.Index  = KNN.getNumericalFeatures(train_X)
        return (((train_X[numerical_features].sub(inp[numerical_features]).abs()) **
p).sum(axis = 1)) ** (1 / p)

    def fit(self, X: pd.DataFrame, y: pd.Series):
        self.train_X = X
        self.train_y = y

    def get_params(self, deep: bool = True) -> Dict[str, Any]:
        return {"k": self.k, "method": self.method, "p": self.p}

    def predict(self, input_X: Union[pd.DataFrame, pd.Series]) -> List[int]:
        """
        Mencari klasifikasi dari data masukan menggunakan algoritma K-Nearest
Neighbors (KNN).
        Fungsi ini mengembalikan kelas yang paling sering muncul di antara K
tetangga terdekat
        berdasarkan jarak yang dihitung menggunakan salah satu dari metode berikut:
            1. Manhattan Distance (manhattan)
            2. Euclidean Distance (euclidean)
            3. Minkowski Distance (minkowski) - Memerlukan tambahan parameter p.
        """
        def classify_single(inp_row: pd.Series) -> str:
            distances: pd.Series = None
            if self.method == "manhattan":
                distances = KNN.calcMinkowskiDist(self.train_X, inp_row, 1)
            elif self.method == "euclidean":
                distances = KNN.calcMinkowskiDist(self.train_X, inp_row, 2)
            elif self.method == "minkowski":
                if self.p == None:
                    raise ValueError("Nilai p kosong.")
                distances = KNN.calcMinkowskiDist(self.train_X, inp_row, self.p)
            else:
                raise ValueError("Method tidak valid.")

            # Cari K tetangga dengan distance terkecil.
            kNearestNeighbour: pd.Series = np.argsort(distances)[:self.k]

            # Cari kelas yang paling sering muncul di antara KNN.
            target: Dict[str, int] =
self.train_y.iloc[kNearestNeighbour].value_counts().to_dict()

            # Kembalikan kelas yang paling sering muncul
            most_common_class: str = max(target, key = target.get)
            return int(most_common_class)
```

```
        # Cek model sudah di fit belum
        if self.train_X is None or self.train_y is None:
            raise ValueError("Model belum di fit.")

        # Cari klasifikasi menggunakan KNN
        if isinstance(input_X, pd.Series):
            return [classify_single(input_X)]
        elif isinstance(input_X, pd.DataFrame):
            return [classify_single(input_X.iloc[i]) for i in range(len(input_X))]
        else:
            raise TypeError("Tipe input bukan merupakan pd.Series ataupun
pd.DataFrame.")

    def save(self, save_path: str) -> None:
        """
        Mengubah data model ke dalam bentuk json dan menyimpannya berdasarkan
        path yang diberikan.
        """
        serialized_model: Dict[str, Any] = {
            "k": self.k,
            "method": self.method,
            "p": self.p,
            "train_X": None if self.train_X is None else self.train_X.to_dict(),
            "train_y": None if self.train_y is None else self.train_y.to_list()
        }

        with open(save_path, "w") as file:
            json.dump(serialized_model, file)

        print(f"Model tersimpan di {save_path}")

    @classmethod
    def load(cls, load_path: str) -> "KNN":
        """
        Membuka file yang berisi data model dan mengembalikan objek kelas KNN
berdasarkan
        data yang telah dimuat.
        """
        try:
            with open(load_path, "r") as file:
                serialized_model: Dict[str, str] = json.load(file)

            knn: "KNN" = cls(
                k = serialized_model["k"],
                method = serialized_model["method"],
                p = serialized_model["p"]
            )
```

```
        knn.train_X = None if serialized_model["train_X"] is None else
pd.DataFrame(serialized_model["train_X"])
        knn.train_y = None if serialized_model["train_y"] is None else
pd.Series(serialized_model["train_y"])

        print(f"Berhasil memuat model dari {load_path}")

        return knn
    except FileNotFoundError:
        raise FileNotFoundError(f"File {load_path} tidak ditemukan.")
```

2. Naive-Bayes

Algoritma Naive Bayes adalah salah satu algoritma *supervised learning* yang memanfaatkan hipotesis atau model. Hasil dari model Naive Bayes adalah peluang kemunculan kelas serta peluang kemunculan nilai atribut jika diberikan kelas tertentu. Peluang kelas dan peluang atribut kategorikal dihitung dari frekuensi kemunculan kelas atau atribut tersebut. Sedangkan, peluang atribut numerik dihitung berdasarkan distribusi *normal*/*gaussian*, yaitu menggunakan *probability density function*. Karena perbedaan proses perhitungan peluang atribut kategorikal dan numerik, proses pelatihan/*training* algoritma Gaussian Naive Bayes dipisah menjadi dua, yaitu pelatihan model kategorikal dan pelatihan model numerik.

Hasil dari model kategorikal adalah peta*/map* yang berisi frekuensi dan peluang setiap kelas dan atribut pada setiap kelas/klasifikasi. Dalam *source code*, peta ini disimpan dalam *field* bernama *frequency* dan *probability*. Algoritma pertama-tama mencari frekuensi setiap kelas dan frekuensi setiap nilai unik dari setiap atribut terhadap setiap kelas. Setelah diperoleh semua frekuensi, algoritma mengkalkulasi peluang atau *probability* setiap kelas dan setiap atribut. Peluang setiap kelas dihitung dengan cara membagi frekuensi kelas dengan jumlah data. Peluang setiap nilai unik dari setiap atribut terhadap setiap kelas dihitung dengan cara membagi frekuensi dari nilai atribut dengan frekuensi kelas yang bersangkutan.

gaussian_naive_bayes.py

NaiveBayes.__trainCategorical()

```
def __trainCategorical(self):
    attr_numerical = self.train_X.select_dtypes(include=float).columns
    attr_categorical = self.train_X.copy().drop(columns=attr_numerical)
    bn_target = self.train_y.copy()

    numData = len(attr_categorical.index)

    self.probability = {'target': {}}
    self.frequency = {'target': {}}

    # maps the frequencies of every unique values of each attribute categorical
```

```
with each unique target label value
        # map structure:
        # attribute1:
        #   attributeValue1:
        #       targetValue1: {frequency}
        #   ...
        # attribute2:
        #   ...
        # targetValue:
        #   targetValue1: {frequency}
        #   ...

        for id in attr_categorical.index:
            # add the frequency of the targetLabel value
            targetValue = str(bn_target[id])

            # TODO un-hardcode target
            if targetValue not in self.frequency['target']:
                self.frequency['target'][targetValue] = 1
            else:
                self.frequency['target'][targetValue] += 1

            for column in attr_categorical:
                column = str(column)

                # put attribute/targetValue in map
                if column not in self.frequency:
                    self.frequency[column] = {}
                    self.probability[column] = {}

                attrValue = str(attr_categorical[column][id])
                if attrValue not in self.frequency[column]:
                    self.frequency[column][attrValue] = {}
                    self.probability[column][attrValue] = {}

                if targetValue not in self.frequency[column][attrValue]:
                    self.frequency[column][attrValue][targetValue] = 1
                    self.probability[column][attrValue][targetValue] = 0
                else:
                    self.frequency[column][attrValue][targetValue] += 1

        # maps the probability of every unique values of each categorical attribute
with each unique target label value
        # map structure:
        # attribute1:
        #   attributeValue1:
        #       targetValue1: {frequency}
        #   ...
        # attribute2:
```

```
    #   ...
    # targetValue:
    #   targetValue1: {frequency}
    #   ...

    for column in self.frequency:
        column = str(column)

        if column == 'target':
            for cat in self.frequency[column]:
                cat = str(cat)

                self.probability[column][cat] = self.frequency[column][cat]/numData
            continue

        for value in self.frequency[column]:
            value = str(value)

            for cat in self.frequency[column][value]:
                self.probability[column][value][cat] =
self.frequency[column][value][cat]/self.frequency['target'][cat]
```

Hasil dari pelatihan model numerik adalah peta/*map* yang berisi rata-rata dan standar deviasi dari setiap atribut pada setiap kelas/klasifikasi. Rata-rata dan standar deviasi ini lah yang akan digunakan untuk menghitung *probability density function* pada proses prediksi. Algoritma pertama-tama memilah data berdasarkan kelas. Kemudian, algoritma menghitung rata-rata dan standar deviasi dari setiap atribut numerik untuk setiap data yang sudah dipilah. Rata-rata dan standar deviasi yang diperoleh kemudian disimpan di dalam sebuah *variable* peta atau *map*. Dalam *source code*, peta ini disimpan dalam *field* bernama *meanMap* dan *sdevMap*.

gaussian_naive_bayes.py

NaiveBayes.__trainNumeric()

```
def __trainNumeric(self):
    # maps the mean of every numerical attributes according to their
target/classification
    # map structure:
    # numAttribute1:
    #   (mean/sdev) targetValue1:
    #   ...
    # numAttribute2:
    #   ...

    attr_numerical = self.train_X.select_dtypes(include=float).columns
    self.meanMap = {}
    self.sdevMap = {}
```

```
    targetValues = self.train_y.unique()
    for attr in attr_numerical:
        attr = str(attr)

        if attr not in self.meanMap:
            self.meanMap[attr] = {}

        if attr not in self.sdevMap:
            self.sdevMap[attr] = {}

        for value in targetValues:
            targetIndex = self.train_y[self.train_y == value].index
            filter = self.train_X.loc[targetIndex, [attr]].to_numpy()

            value = str(value)
            self.meanMap[attr][value] = np.mean(filter)
            self.sdevMap[attr][value] = np.std(filter)
```

Untuk memprediksi kelas suatu *unseen data*, algoritma akan menghitung peluang data terhadap setiap kelas, lalu mengembalikan kelas yang memiliki peluang paling besar. Peluang suatu kelas dihitung dengan cara mengalikan peluang kelas tersebut dengan peluang setiap atribut.

gaussian_naive_bayes.py

NaiveBayes.predict_single(pandas.Series)

```
    def predict_single(self, row: pd.Series) -> str:
        attr_numerical = self.meanMap.keys()
        attr_categorical = self.probability.keys()

        max_probability = -1
        max_target = None

        for targetVal in self.probability['attack_cat'].keys():
            probability_val = self.__probabilityTarget(targetAttr=targetVal)
            for attr, value in row.items():
                attr = str(attr)

                if attr in attr_numerical:
                    probability_val *= self.__probabilityNumeric(attrValue=value,
attribute=attr, targetValue=targetVal)
                elif attr in attr_categorical:
                    value = str(value)
                    probability_val *= self.__probabilityCategorical(attrValue=value,
attribute=attr, targetValue=targetVal)
                # else attribute not in train set, skip
```

```
            if probability_val > max_probability:
                max_probability = probability_val
                max_target = targetVal

        return max_target
```

NaiveBayes.predict(pandas.DataFrame)

```
    def predict(self, test_X: pd.DataFrame) -> List[int]:
        if isinstance(test_X, pd.Series):
            return [self.predict_single(test_X)]
        elif isinstance(test_X, pd.DataFrame):
            return [self.predict_single(test_X.iloc[i]) for i in range(len(test_X))]
        else:
            raise TypeError("Tipe input bukan merupakan pd.Series ataupun
pd.DataFrame.")
```

Nilai peluang suatu kelas atau atribut kategorikal diperoleh dari peta peluang yang telah dibentuk dari hasil pelatihan model kategorikal.

gaussian_naive_bayes.py

NaiveBayes.__probabilityCategorical

```
def __probabilityCategorical(self, attrValue, attribute, targetValue):
        try:
            prob = self.probability[attribute][attrValue][targetValue]
        except KeyError:
            prob = 0
        return prob
```

NaiveBayes.__probabilityTarget

```
def __probabilityTarget(self, targetAttr):
        try:
            prob = self.probability['attack_cat'][targetAttr]
        except KeyError:
            prob = 0
        return prob
```

Sedangkan nilai peluang suatu atribut kategorikal diperoleh dengan menghitung *probability density function* dari atribut tersebut. Peluang dihitung berdasarkan rumus berikut:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

source: https://en.wikipedia.org/wiki/Normal_distribution

Nilai rata-rata dan standar deviasi diperoleh dari peta rata-rata dan standar deviasi yang telah dibentuk dari hasil pelatihan model numerik.

gaussian_naive_bayes.py

NaiveBayes.__probabilityNumeric

```
def __probabilityNumeric(self, attrValue, attribute, targetValue):
    def normalDistribution(x, mean, sdev):
        pi = np.pi

        f_x = (1/(np.sqrt(2*pi) * sdev)) * np.exp(-0.5 * (((x-mean)/sdev)**2))
        return f_x

    try:
        mean = self.meanMap[attribute][targetValue]
        sdev = self.sdevMap[attribute][targetValue]
    except KeyError:
        mean = 0
        sdev = 0

    if (mean == 0 or sdev == 0):
        return 0

    prob = normalDistribution(attrValue, mean, sdev)
    return prob
```

Implementasi algoritma juga memiliki fitur penyimpanan dan pemuatan model ke dalam file JSON.

gaussian_naive_bayes.py

NaiveBayes.save

```
def save(self, save_path: str) -> None:
    """
    Mengubah data model ke dalam bentuk json dan menyimpannya berdasarkan
    path yang diberikan.
    """
    serialized_model: Dict[str, Any] = {
        "mean_map": None if self.meanMap is None else self.meanMap,
        "sdev_map": None if self.sdevMap is None else self.sdevMap,
        "frequency_map": None if self.frequency is None else self.frequency,
        "probability_map": None if self.probability is None else self.probability
    }

    for items in serialized_model:
        print(items)
    print(json.dumps(serialized_model))
```

```
        with open(save_path, "w") as file:
            json.dump(serialized_model, file, indent=6)

        print(f"Model Naive Bayes tersimpan di {save_path}")
```

NaiveBayes.load

```
@classmethod
    def load(cls, load_path: str) -> "NaiveBayes":
        """
        Membuka file yang berisi data model dan mengembalikan objek kelas NaiveBayes berdasarkan
        data yang telah dimuat.
        """
        try:
            with open(load_path, "r") as file:
                serialized_model: Dict[str, str] = json.load(file)

            nb: "NaiveBayes" = cls()

            nb.meanMap = None if serialized_model["mean_map"] is None else dict(serialized_model["mean_map"])
            nb.sdevMap = None if serialized_model["sdev_map"] is None else dict(serialized_model["sdev_map"])
            nb.frequency = None if serialized_model["frequency_map"] is None else dict(serialized_model["frequency_map"])
            nb.probability = None if serialized_model["probability_map"] is None else dict(serialized_model["probability_map"])

            print(f"Berhasil memuat model dari {load_path}")

            return nb
        except FileNotFoundError:
            raise FileNotFoundError(f"File {load_path} tidak ditemukan.")
```

3. ID3

Algoritma ID3 (Iterative Dichotomizer 3) adalah sebuah algoritma supervised learning yang memodelkan data menjadi sebuah decision tree.

Persamaan yang diperlukan untuk algoritma ID3 adalah sebagai berikut:

a. Entropi

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

b. Information Gain

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Berikut adalah implementasi dari ID3

id3.py

```
import numpy as np
import pandas as pd
from collections import Counter

class ID3DecisionTree:
    def __init__(self):
        self.tree = None

    def entropy(self, y):
        # fungsi untuk menghitung entropi
        cnts = Counter(y)
        total = len(y)
        return -sum((cnt/total) * np.log2(cnt/total) for cnt in cnts.values())

    def information_gain(self, X_col, y):
        #hitung information gain
        entropyS = self.entropy(y)
        vals, cnts = np.unique(X_col, return_counts=True)
        total = len(X_col)
        weighted_entropy = sum((cnts[i]/total)*self.entropy(y[X_col==value]) for
i,value in enumerate(vals))
        return entropyS - weighted_entropy

    def best_split(self, X, y):
        #pilih feature terbaik saat split
        gains = [(col, self.information_gain(X[col], y)) for col in X.columns]
        return max(gains, key=lambda x: x[1])

    def fit(self, X, y):
        #train data menggunakan ID3
        if len(np.unique(y)) == 1:
            return y.iloc[0]  #NODE LEAF

        if X.empty:
            return y.mode()[0]  #Kelas mayoritas jika fiturnya habis
        best_feature, _ = self.best_split(X, y)
        tree = {best_feature: {}}

        for value in np.unique(X[best_feature]):
            sub_X = X[X[best_feature] == value].drop(columns=[best_feature])
```

```
            sub_y = y[X[best_feature] == value]
            tree[best_feature][value] = self.fit(sub_X, sub_y)

        self.tree = tree
        return tree

    def predict_instance(self, instance, tree):
        #prediksi stu buah instance
        if not isinstance(tree, dict):
            return tree

        feature = next(iter(tree))
        value = instance[feature]
        subtree = tree[feature].get(value, None)

        if subtree is None:
            return None

        return self.predict_instance(instance, subtree)

    def predict(self, X):
        # prediksi banyak instance
        return X.apply(lambda row: self.predict_instance(row, self.tree), axis=1)
```

Data Cleaning dan Preprocessing

1. Data Cleaning

Proses *data cleaning* dilakukan untuk membersihkan dataset dari data-data yang tidak valid maupun data-data yang "kotor", sehingga hanya data berkualitas lah yang akan digunakan untuk melakukan training model. Data cleaning yang kami lakukan terdiri dari 5 langkah: Handling Missing Data, Dealing with Outliers, Data Validation, Removing Duplicates, dan Feature Engineering.

- Handling Missing Data

  Pada langkah Handling Missing Data, dilakukan imputasi pada data-data yang memiliki nilai yang hilang pada satu atau lebih atribut. Data yang memiliki *missing values* jumlahnya ternyata sangat banyak (87,8% dari train set memiliki *missing values*), sehingga tidak mungkin dilakukan penghapusan pada data-data tersebut

(*deletion of missing data*). Sehingga, kami memutuskan untuk melakukan imputasi pada data-data tersebut dengan ketentuan:

- Untuk fitur kategorikal, nilai missing value diganti dengan nilai modus agar distribusi keseluruhan datanya cenderung tidak berubah.
- Untuk fitur numerik, nilai missing value diganti dengan nilai median. Kita memilih untuk mengganti dengan nilai median karena bila kita misalnya menggunakan nilai *mean*, nilai tersebut bisa menjadi terlalu ekstrim tergantung dari seberapa jauh selisih nilai data outlier dengan data aslinya.

- Dealing with Outliers

  Pada langkah Dealing with Outliers, dilakukan imputasi pada data-data yang dianggap outlier terhadap keseluruhan data dengan menggunakan nilai IQR, yaitu suatu data dianggap outlier jika nilainya lebih kecil dari Q1 - 1,5*IQR atau lebih besar dari Q3 + 1,5*IQR. Berdasarkan ketentuan ini, data yang dianggap outlier jumlahnya ternyata sangat banyak (82,15% dari train set dan 82,28% dari validation set) maka outlier ini tidak mungkin dihapus begitu saja. Kami memutuskan untuk menangani outlier dengan cara sama seperti yang telah dilakukan pada langkah sebelumnya, yaitu mengganti nilai outlier dengan nilai median dari atribut yang bersesuaian, dengan alasan sama seperti pada langkah sebelumnya.

- Removing Duplicates

  Pada langkah Removing Duplicates, dilakukan pengecekan terlebih dahulu berapa banyak data yang sebenarnya duplikat dari data yang telah ada. Cara menentukan apakah suatu data adalah duplikat atau bukan ditentukan dengan mengecek apakah semua nilai atribut dari kedua data sama kecuali untuk atribut ‘ID’, karena atribut ‘ID’ hanya menandakan urutan entri data dan tidak digunakan oleh model untuk memprediksi. Berdasarkan ketentuan tersebut, ternyata ditemukan sebanyak 20,45% dari training set dan 14,67% dari validation set adalah data duplikat. Walaupun jumlah data duplikat ini tergolong relatif banyak, kita tidak mungkin menanganinya dengan cara yang sama seperti yang telah kita lakukan sebelumnya yaitu imputasi karena data-data ini adalah data-data yang benar-benar sama dari yang sudah ada, sehingga data tersebut tidak akan menambah “insight” baru dan sebaiknya dihilangkan saja karena bila tidak dihilangkan dapat membuat model kita terlalu fokus pada data yang memiliki banyak duplikat.

- Feature Engineering

  Pada langkah Feature Engineering, dilakukan pemilihan fitur yaitu menghilangkan fitur-fitur dari data training yang tidak berkontribusi banyak terhadap kemampuan model untuk memprediksi dengan baik. Cara pemilihan fitur ini menggunakan library yaitu RandomForest dari sklearn, dimana model RandomForest dari sklearn ini akan melakukan training pada model *random forest* yang dimilikinya (menggunakan dataset x dan y) dan menghitung skor *importance* dari tiap atribut. Bila skor importance dibawah threshold, maka fitur itu akan dihilangkan pada tahap *transform* model. Dengan menggunakan cara ini, kami melakukan feature engineering

tanpa menentukan fitur apa saja yang harus dipilih maupun berapa fitur yang harus dipilih. Namun karena RandomForest memerlukan semua nilai atribut berupa numerik, maka kami memutuskan untuk melakukan langkah Feature Engineering ini bersama dengan *data preprocessing*.

2. Data Preprocessing

Data preprocessing dilakukan untuk mengolah dataset menjadi data yang siap pakai untuk tahap training. Langkah ini perlu dilakukan karena misalnya beberapa model *supervised learning* hanya dapat menerima data yang semua atributnya numerik, atau beberapa model mengasumsikan data training berdistribusi normal, atau jumlah fitur dataset terlalu banyak sehingga kompleksitas waktu tahap training terlalu besar, atau banyak fitur noise, dan masih banyak lagi. Data preprocessing yang kami lakukan terdiri dari 5 langkah:

- Feature Scaling

  Pada langkah Feature Scaling, nilai atribut numerik pada dataset distandardisasi sehingga semua atribut numerik memiliki range nilai yang sama. Hal ini dilakukan karena banyak model yang bergantung pada besar kecilnya nilai numerik fitur, sehingga bila tidak ditangani fitur yang memiliki *range* nilai yang besar akan berdampak jauh lebih besar pada model daripada fitur-fitur dengan *range* nilai yang kecil. Untuk langkah Feature Scaling ini, kami menggunakan Min-Max Scaling karena pada tahap ini distribusi data pada dataset masih belum diolah menjadi distribusi normal. Pada tahap modeling kami juga tidak memakai model-model yang mengasumsikan dataset terdistribusi normal (misalnya SVM, Logistic Regression, dll.) sehingga Min-Max Scaling kami rasa aman untuk dipilih.

- Feature Encoding

  Pada langkah Feature Encoding, semua atribut kategorikal pada dataset diubah menjadi bentuk numerik agar dapat digunakan oleh model. Untuk langkah ini, kami menggunakan One-Hot Encoding karena berdasarkan deskripsi yang terdapat pada file UNSW-NB15, fitur-fitur kategorikal yang ada (attack_cat, proto, state, service) sifatnya bukanlah ordinal sehingga tidak cocok untuk menggunakan Label Encoding. Fitur target yaitu attack_cat juga kategorikal dan sifatnya bukanlah ordinal, sehingga tidak cocok pula untuk menggunakan Mean Encoding.

- Handling Imbalanced Dataset

  Pada langkah Handling Imbalanced Dataset, kami melakukan undersampling untuk membuat semua kelas memiliki jumlah data yang sama pada dataset sehingga tidak ada kelas yang terlalu mendominasi pada saat dilakukan training. Kami memilih untuk melakukan undersampling karena jumlah data yang ada pada training set banyak (setelah dilakukan penghapusan data duplikat pun masih berjumlah 111591 data), sehingga kami rasa undersampling lebih cocok untuk mengurangi jumlah data untuk training agar waktu training yang diperlukan lebih singkat (bila melakukan

oversampling maka training dapat memerlukan waktu hingga puluhan menit bahkan berjam-jam) sekaligus memastikan tidak ada kelas yang mendominasi data.

- Data Normalization

  Pada langkah Data Normalization, kami mengolah dataset agar menjadi berdistribusi normal. Hal ini perlu dilakukan karena cara kerja dari beberapa model *supervised learning* mengasumsikan data training berdistribusi normal, sehingga bila data training yang digunakan nyatanya tidak berdistribusi normal maka akan berpengaruh buruk ke performa model-model tersebut. Untuk langkah ini, kami memilih untuk menggunakan model QuantileTransformer dengan pilihan output distribusi yaitu distribusi normal.

- Dimensionality Reduction

  Pada langkah Dimensionality Reduction, kami mengolah dataset dengan cara "menggabungkan" fitur-fitur numerik menjadi beberapa fitur numerik baru yang merepresentasikan fitur-fitur numerik awal, dan jumlahnya lebih sedikit dari fitur awal. Tujuan dari dimensionality reduction adalah untuk mengurangi jumlah fitur dengan cara membuat fitur baru yang merupakan "gabungan" dari fitur-fitur numerik awal yang masing-masing diberi bobot, sehingga dihasilkan nilai dari fitur baru ini. Untuk langkah ini, kami menggunakan PCA karena PCA merupakan metode yang paling sering digunakan untuk dimensionality reduction dan metode-metode lain menurut kami kurang cocok untuk kasus tugas besar 2 ini (misalnya, kita tidak memerlukan visualisasi yang disediakan t-SNE dan kita juga tidak menggunakan deep learning untuk mengekstrak data non-linear yang disediakan *autoencoder*).

Analisis Hasil Prediksi

1. Perbandingan Hasil Prediksi Algoritma *From Scratch* dengan Algoritma Scikit-Learn

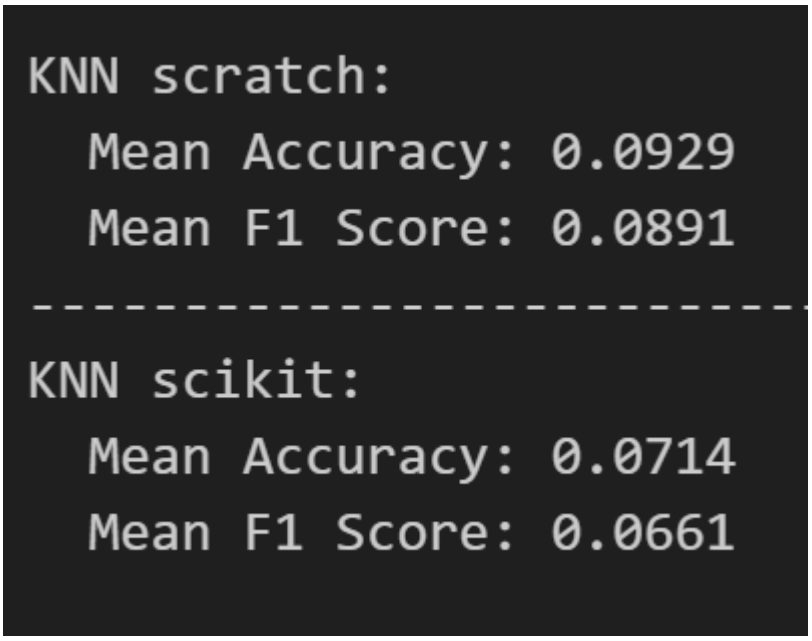
Pada tahap ini, algoritma yang telah diimplementasikan dari awal (*from scratch*) akan dibandingkan dengan algoritma yang disediakan oleh *library* scikit-learn. Metrik yang akan digunakan untuk melakukan perbandingan adalah *accuracy* (akurasi) dan F1 Score, yang dapat dihitung menggunakan rumus berikut,

$$Accuracy = \frac{Jumlah\ Prediksi\ Benar}{Jumlah\ Total\ Data},$$

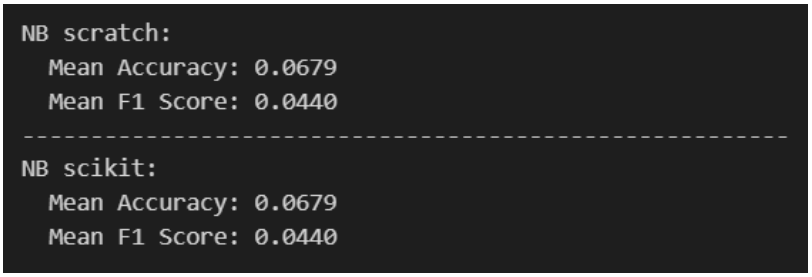
$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}.$$

Untuk memudahkan proses perbandingan, algoritma penghitungan akurasi dan F1-Score akan diambil dari pustaka scikit-learn.

a. KNN



b. Naive-Bayes



c. ID3

2. Insight yang Diperoleh dari Hasil Perbandingan

Untuk KNN, nilai akurasi dan f1 score algoritma *from scratch* lebih tinggi dibandingkan algoritma *scikit-learn*, hal ini mungkin karena algoritma dari *scikit-learn* lebih teroptimisasi untuk dataset yang lebih besar dan lebih baik dalam menghindari overfitting. Algoritma *scikit-learn* menggunakan struktur data yang lebih efisien seperti KD-Trees dan Ball Trees, yang mempercepat proses pencarian tetangga terdekat pada dataset besar. Sedangkan, algoritma KNN *from scratch* jauh lebih sederhana dan mengandalkan perhitungan jarak secara langsung antara semua pasangan data yang dapat menyebabkan overfitting pada dataset kecil dan menghasilkan akurasi yang lebih tinggi.

Dapat dilihat bahwa algoritma Naive Bayes yang dibuat *from scratch* memiliki akurasi yang sama dengan algoritma yang diambil dari pustaka *scikit-learn*. Hal ini mungkin dikarenakan tidak adanya perbedaan algoritma maupun *hyperparameter* antara algoritma *from scratch* dan algoritma *scikit-learn*. Berbeda dengan KNN, Naive Bayes tidak menerima parameter seperti k-jumlah-neighbour. Algoritma *scikit-learn* juga mungkin tidak melakukan *preprocessing* lebih lanjut terhadap data yang diberikan.

Lampiran

Pranala repository GitHub: Tubes-AI-2-K9

Tabel pembagian tugas:

NIM	Tugas
13522002	Model NB, Validasi NB
13522007	Data Cleaning, Data Preprocessing
13522036	Model ID3, Validasi ID3
13522056	Model KNN, Validasi KNN