

Laporan Tugas Besar 2 IF3170 Inteligensi Artifisial
Implementasi Algoritma Pembelajaran Mesin
Semester I Tahun 2024/2025



Disusun oleh:

1. Maulana Muhamad Susetyo 13522127
2. Andi Marihot Sitorus 13522138
3. Muhammad Dzaki Arta 13522149
4. Muhammad Rasheed Qais Tandjung 13522158

INSTITUT TEKNOLOGI BANDUNG

2024

Implementasi Algoritma

1. K-Nearest Neighbor (KNN)

Algoritma K-Nearest Neighbor (KNN) adalah salah satu metode klasifikasi yang paling sederhana dan intuitif dalam machine learning. KNN bekerja dengan cara mengklasifikasikan data baru berdasarkan kedekatannya dengan data yang ada di dalam dataset. Kelas KNN dalam implementasi ini dirancang untuk menyediakan berbagai fungsi yang mendukung proses pelatihan model, prediksi, serta penyimpanan dan pemuatan model. Kelas ini memiliki beberapa fungsi utama yang akan dijelaskan secara rinci berikut.

Fungsi fit

Fungsi ini digunakan untuk melatih model dengan data latih. Fungsi ini menerima dua parameter: `X_train`, yang merupakan fitur atau atribut data latih, dan `y_train`, yang merupakan label kelas dari data latih. Fungsi ini akan menyimpan data latih dalam variabel instansi sehingga dapat digunakan dalam fungsi lain, seperti prediksi dan perhitungan jarak.

```
def fit(self, X_train, y_train):  
    self.X_train = X_train  
    self.y_train = y_train
```

Fungsi predict

Fungsi ini menerima data uji (`X_test`) dan memprediksi kelas untuk setiap baris data uji. Untuk setiap baris data uji, fungsi ini akan memanggil fungsi `get_neighbours` untuk mencari tetangga terdekat dari data uji berdasarkan metrik jarak yang dipilih (seperti Euclidean, Manhattan, atau Minkowski). Setelah mendapatkan tetangga terdekat, fungsi ini akan menghitung mayoritas kelas yang ada di antara tetangga tersebut dan mengembalikan kelas yang paling sering muncul sebagai prediksi.

```
def predict(self, X_test):  
    preds = []  
    for test_row in X_test:  
        nearest_neighbours = self.get_neighbours(test_row)  
        unique, counts = np.unique(nearest_neighbours,  
return_counts=True)  
        majority = unique[np.argmax(counts)]  
        preds.append(majority)  
    return np.array(preds)
```

Fungsi get_neighbours

Fungsi ini digunakan untuk mencari tetangga terdekat dari suatu data uji. Fungsi ini menghitung jarak antara baris data uji dengan setiap baris data latih menggunakan berbagai metrik yang dapat dipilih: Euclidean, Manhattan, atau Minkowski. Setelah menghitung jarak, fungsi ini mengurutkan jarak tersebut dan memilih `k` tetangga terdekat

untuk data uji. Fungsi ini kemudian mengembalikan kelas dari tetangga-tetangga tersebut untuk digunakan dalam perhitungan mayoritas kelas pada fungsi predict.

```
def get_neighbours(self, test_row):
    distances = list()

    for (train_row, train_class) in zip(self.X_train,
self.y_train):
        if self.metric == 'euclidean':
            dist =
np.sqrt(np.sum((train_row-test_row)**2))
        elif self.metric == 'manhattan':
            np.sum(np.abs(train_row-test_row))
        elif self.metric == 'minkowski':
            np.sum(np.abs(train_row-test_row)**self.p)**(1/self.p)
        else:
            raise NameError('Parameter metrics should be
euclidean, manhattan and minkowski')
        distances.append((dist, train_class))

    distances.sort(key=lambda x: x[0])

    neighbours = list()
    for i in range(self.k):
        neighbours.append(distances[i][1])

    return neighbours
```

Fungsi save_to_file

Fungsi ini digunakan untuk menyimpan model KNN yang telah dilatih ke dalam sebuah file. Fungsi ini menyimpan informasi tentang nilai k (jumlah tetangga yang digunakan), metric (jenis metrik jarak), serta data latih (X_train dan y_train). Data tersebut disimpan dalam format JSON agar mudah diakses kembali di lain waktu.

```
def save_to_file(self, filename):
    model_data = {
        'k': self.k,
        'metric': self.metric,
        'X_train': self.X_train.tolist(),
        'y_train': self.y_train.tolist()
    }
    with open(filename, 'w') as f:
        json.dump(model_data, f)
    print(f"Model saved to {filename}")
```

Fungsi load_from_file

Fungsi ini digunakan untuk memuat model KNN yang telah disimpan sebelumnya dari sebuah file. Fungsi ini akan membaca file JSON yang berisi data model, kemudian

mengembalikan nilai k, metric, dan data latih yang diperlukan untuk melanjutkan prediksi dengan model yang sama.

```
def load_from_file(self, filename):
    with open(filename, 'r') as f:
        model_data = json.load(f)
        self.k = model_data['k']
        self.metric = model_data['metric']
        self.X_train = np.array(model_data['X_train'])
        self.y_train = np.array(model_data['y_train'])
        print(f"Model loaded from {filename}")
    return self
```

2. Naive Bayes

Algoritma *Naive Bayes* merupakan salah satu metode untuk perancangan model pada *machine learning*, yang menggunakan pendekatan probabilistik untuk melakukan prediksi kelas, yaitu dengan menentukan kelas sebuah input berdasarkan probabilitas yang terobservasi dari training data.

Implementasi *Naive Bayes* pada tugas besar ini merupakan tipe *Gaussian Naive Bayes*, yaitu versi *Naive Bayes* yang fitur numeriknya diasumsikan mengikuti distribusi Gaussian.

Salah satu aspek utama pada algoritma *Naive Bayes* adalah mengasumsikan independensi antar fitur (naive). Hal ini dapat menjadi aspek yang memperkuat atau memperlemah model dalam prediksi, tergantung data yang dimiliki.

Fungsi fit

```
def fit(self, X_train, y_train):
    self.classes = np.unique(y_train)

    for c in self.classes:
        X_c = X_train[y_train == c]

        self.class_mean[c] = X_c.mean(axis=0)
        self.class_var[c] = X_c.var(axis=0) + 1e-9
        self.class_prior[c] = X_c.shape[0] /
X_train.shape[0]
```

Fungsi ini akan menerima *X_train* dan *y_train*, lalu untuk setiap kelas akan menetapkan mean, variansi, dan probabilitas dari kelas tersebut.

Fungsi predict

```
def predict(self, X_test):
    preds = []
    for test_row in X_test:
```

```

        posteriors = []
        for c in self.classes:
            prior_log = np.log(self.class_prior[c])
            likelihood_log =
np.sum(np.log(self._gaussian_likelihood(test_row, c)))
            posterior = prior_log + likelihood_log
            posteriors.append(posterior)
        preds.append(self.classes[np.argmax(posteriors)])

    return np.array(preds)

```

Fungsi prediksi pada implementasi Gaussian Naive Bayes akan menghitung untuk setiap kelas c nilai posterior, yaitu penjumlahan prior probability dari kelas c dengan likelihood untuk setiap atribut sebuah baris pada setiap fitur, yaitu $L(\text{atribut} | c)$.

Fungsi prediksi lalu akan mengembalikan kelas dengan nilai posterior tertinggi.

Fungsi `_gaussian_likelihood`

```

def _gaussian_likelihood(self, x, class_label):
    mean = self.class_mean[class_label]
    var = self.class_var[class_label]

    numerator = np.exp(-0.5 * ((x - mean) ** 2) / var)
    denominator = np.sqrt(2 * np.pi * var)

    return numerator / denominator

```

Fungsi ini akan menghitung likelihood sebuah nilai x pada *gaussian distribution* yang diberikan oleh mean dan var yang sudah disimpan model untuk kelas `class_label`.

Fungsi `save_to_file`

```

def save_to_file(self, filename):
    with open(filename, 'wb') as f:
        pickle.dump(self, f)

    @staticmethod
    def load_from_file(filename):
        with open(filename, 'rb') as f:
            return pickle.load(f)

```

3. ID3

Kelas ID3 memiliki sebuah parameter yaitu `max_depth` untuk kedalaman maksimalnya. Dalam kelas ID3 terdapat fungsi `fit` dan `predict` yang akan digunakan pada pipeline. Terdapat juga fungsi-fungsi yang digunakan untuk membuat decision tree menggunakan algoritma ID3.

Fungsi entropy

Menghitung entropy variabel target

```
def entropy(self, y):
    _, y = np.unique(y, return_inverse=True)
    counts = np.bincount(y)
    probabilities = counts / len(y)
    return -np.sum(probabilities * np.log2(probabilities +
1e-9))

    self.y_train = y_train
```

Fungsi information_gain

Menghitung information gain dari sebuah kolom

```
def information_gain(self, X, y, feature_index,
threshold=None):
    """Calculate information gain for a feature."""
    total_entropy = self.entropy(y)

    # Split on threshold if provided
    if threshold is None:
        values, counts = np.unique(X[:, feature_index],
return_counts=True)
        weighted_entropy = 0
        for i, value in enumerate(values):
            subset_y = y[X[:, feature_index] == value]
            weighted_entropy += (counts[i] / len(y)) *
self.entropy(subset_y)
    else:
        # Binary split for numerical feature
        left_indices = X[:, feature_index] <= threshold
        right_indices = X[:, feature_index] > threshold

        left_entropy = self.entropy(y[left_indices]) if
np.any(left_indices) else 0
        right_entropy = self.entropy(y[right_indices]) if
np.any(right_indices) else 0

        left_weight = np.sum(left_indices) / len(y)
        right_weight = np.sum(right_indices) / len(y)

        weighted_entropy = left_weight * left_entropy +
right_weight * right_entropy

    return total_entropy - weighted_entropy
```

Fungsi best_split

Mencari kolom dengan information gain terbaik dan menambahkan threshold jika kolom merupakan numerik

```

def best_split(self, X, y):
    best_feature = None
    best_gain = -np.inf
    best_threshold = None

    for feature_index in range(X.shape[1]):
        if np.issubdtype(X[:, feature_index].dtype,
np.number): # Numerical feature
            thresholds = np.unique(X[:, feature_index])
            for threshold in thresholds:
                gain = self.information_gain(X, y,
feature_index, threshold)
                if gain > best_gain:
                    best_gain = gain
                    best_feature = feature_index
                    best_threshold = threshold
            else: # Categorical feature
                gain = self.information_gain(X, y,
feature_index)
                if gain > best_gain:
                    best_gain = gain
                    best_feature = feature_index
                    best_threshold = None

    return best_feature, best_threshold

```

Fungsi build_tree

Membuat decision tree menggunakan fungsi-fungsi yang telah dibuat

```

def build_tree(self, X, y, feature_names, depth=0):
    # Stopping condition
    if len(np.unique(y)) == 1 or (self.max_depth is not
None and depth >= self.max_depth):
        return np.bincount(y).argmax()

    if X.shape[1] == 0:
        return np.bincount(y).argmax()

    # Find the best feature to split on
    best_feature_index, best_threshold =
self.best_split(X, y)
    best_feature_name = feature_names[best_feature_index]

    if best_threshold is not None:
        best_feature_name = f"{best_feature_name} <=
{best_threshold}"

    tree = {best_feature_name: {}}

```

```

        # Handle splitting for numerical or categorical
features
        if best_threshold is None:
            feature_values = np.unique(X[:,
best_feature_index])
            for value in feature_values:
                indices = X[:, best_feature_index] == value
                subtree = self.build_tree(
                    np.delete(X[indices], best_feature_index,
axis=1),
                    y[indices],
                    [name for i, name in
enumerate(feature_names) if i != best_feature_index],
                    depth + 1
                )
                tree[best_feature_name][value] = subtree
            else:
                # Numerical split: create <= threshold and >
threshold branches
                left_indices = X[:, best_feature_index] <=
best_threshold
                right_indices = X[:, best_feature_index] >
best_threshold

                subtree_left = self.build_tree(
                    np.delete(X[left_indices], best_feature_index,
axis=1),
                    y[left_indices],
                    [name for i, name in enumerate(feature_names)
if i != best_feature_index],
                    depth + 1
                )
                subtree_right = self.build_tree(
                    np.delete(X[right_indices],
best_feature_index, axis=1),
                    y[right_indices],
                    [name for i, name in enumerate(feature_names)
if i != best_feature_index],
                    depth + 1
                )

                tree[best_feature_name]["Yes"] = subtree_left
                tree[best_feature_name]["No"] = subtree_right

        return tree

```

Fungsi fit

Menggunakan data training untuk membuat decision tree


```

def fit(self, X, y):
    if isinstance(X, pd.DataFrame):
        feature_names = X.columns.tolist()
    else:
        feature_names = [str(i) for i in
range(X.shape[1])]

    X = np.array(X)
    y = np.array(y)
    self.tree = self.build_tree(X, y, feature_names)
    return self

```

Fungsi predict

Menggunakan decision tree untuk menentukan nilai dari variabel target

```

def predict_instance(self, instance, tree):
    if not isinstance(tree, dict):
        return tree
    feature = next(iter(tree))
    feature_name, threshold = feature.split(" <= ") if " <=
" in feature else (feature, None)
    feature_value = instance[feature_name] if
isinstance(instance, pd.Series) else
instance[int(feature_name)]

    if threshold is not None:
        threshold = float(threshold)
        branch = "Yes" if feature_value <= threshold else
"No"
    else:
        branch = feature_value

    if branch not in tree[feature]:
        return None # Handle unseen feature values
    return self.predict_instance(instance,
tree[feature][branch])

def predict(self, X):
    if isinstance(X, pd.DataFrame):
        predictions = [self.predict_instance(row,
self.tree) for _, row in X.iterrows()]
    else:
        predictions = [self.predict_instance(row,
self.tree) for row in X]
    return np.array(predictions)

```

Fungsi save_to_file

```
def save_to_file(self, filename):  
    if self.tree is None:  
        raise ValueError("Model is not trained yet.")  
    with open(filename, 'w') as f:  
        json.dump(self.tree, f)  
    print(f"Model saved to {filename}")
```

Fungsi load_from_file

```
def load_from_file(self, filename):  
    with open(filename, 'r') as f:  
        self.tree = json.load(f)  
    print(f"Model loaded from {filename}")  
    return self
```

Data Cleaning dan Preprocessing

1. Cleaning

Berikut adalah fungsi `clean_data` yang melakukan pembersihan data sebelum dilakukan preprocessing

```
def clean_data(df_real):  
  
    df = df_real.copy()  
  
    df = gk_lama_imputer(df)  
    df = remove_outliers(df)  
    df = remove_duplicates(df)  
    df = remove_columns(df)  
  
    return df
```

fungsi `gk_lama_imputer` mengganti nilai null dengan nilai:

- median pada data numerikal
- modus pada data kategorikal

```
def gk_lama_imputer(df):  
  
    numeric_imputer = SimpleImputer(strategy='mean')  
    categorical_imputer =  
SimpleImputer(strategy='most_frequent')  
  
    df[Numerical_Columns] =  
numeric_imputer.fit_transform(df[Numerical_Columns])  
    df[Categorical_Columns] =  
categorical_imputer.fit_transform(df[Categorical_Columns])  
  
    return df
```

fungsi `remove_outliers` membuang nilai yang tidak sesuai dari nilai biner dan 99th percentile pada data numerik yang memiliki nilai diluar batas normal (ditentukan oleh penulis)

```
def remove_outliers_num(df_real, num_cols, lower_percentile=0,  
upper_percentile=0.99):  
  
    df = df_real.copy()  
  
    mask = pd.Series(True, index=df.index)  
  
    for col in num_cols:  
        lower_limit = df[col].quantile(lower_percentile)  
        upper_limit = df[col].quantile(upper_percentile)  
        mask &= ((df[col].isna()) | ((df[col] >= lower_limit) &
```

```

(df[col] <= upper_limit)))

    return df[mask]

def remove_outliers_cat(df_real, cat_cols):

    df = df_real.copy()

    for col in cat_cols:
        most_common = df[col].value_counts().nlargest(2).index
        df_cleaned = df[df[col].isin(most_common)]

    return df

def remove_outliers(df_real, rightmost_percent=0.01):

    df = df_real.copy()

    rightmost_columns = [
        "trans_depth", "response_body_len", "sbytes", "dbytes",
"sloss", "dloss", "sjit", "djit", "sinpkt", "dinpkt"
    ]
    most_common_columns = ["is_ftp_login", "swin", "dwin"]

    df_cleaned_col = remove_outliers_cat(df,
most_common_columns)

    df_cleaned = remove_outliers_num(df_cleaned_col,
rightmost_columns)

    return df_cleaned

```

Fungsi `remove_duplicates` menghapus row yang sama persis

```

def remove_duplicates(df):
    result = df.drop_duplicates()
    return df

```

Fungsi `remove_column` menghapus kolom yang redundan/tidak diperlukan. Pada kasus ini, setelah melakukan eksperimen, ditemukan nilai akurasi lebih tinggi jika kolom dengan korelasi tinggi TIDAK dihapus.

```

def remove_columns(df_real):
    df = df_real.copy()

    # abs_corr = corr.abs()
    # upper_triangle = np.triu(np.ones(abs_corr.shape), k=1)
    # high_corr_pairs = [
    #     (abs_corr.index[i], abs_corr.columns[j])

```

```

#     for i, j in zip(*np.where((abs_corr > 0.9) &
(upper_triangle == 1)))
# ]

# groups = []
# seen = set()

# for f1, f2 in high_corr_pairs:
#     if f1 not in seen and f2 not in seen:
#         groups.append({f1, f2})
#         seen.update({f1, f2})
#     else:
#         for group in groups:
#             if f1 in group or f2 in group:
#                 group.update({f1, f2})
#                 seen.update({f1, f2})
#                 break

# groups = [list(group) for group in groups]

# target_correlations = abs_corr['attack_cat']

# columns_to_drop = []

# for group in groups:
#     lowest_corr_column = target_correlations[group].idxmin()
#     columns_to_drop.append(lowest_corr_column)

# df = df.drop(columns=columns_to_drop)

# columns_to_keep = []

# for group in groups:
#     highest_corr_column =
target_correlations[group].idxmax()
#     columns_to_keep.append(highest_corr_column)

# ungrouped_columns = [
#     col for col in abs_corr.columns if col not in seen and
col != 'attack_cat'
# ]
# columns_to_keep.extend(ungrouped_columns)
# columns_to_keep.append('attack_cat')

# df = df[columns_to_keep]

return df.drop('label', axis = 1)

```

2. Preprocessing

Berikut adalah pipeline untuk mendapatkan prediksi dari model

```
def pipeliner(model):
    train_set_clean = clean_data(train_set)
    val_set_clean = clean_data(val_set)

    Categorical_Columns_for_Pipelining =
train_set_clean.select_dtypes(include=['object',
'bool']).columns.tolist()
    Numerical_Columns_for_Pipelining =
train_set_clean.select_dtypes(include=['int64','float64']).columns.tolist()

    Categorical_Columns_for_Pipelining = [col for col in
Categorical_Columns_for_Pipelining if col not in ["attack_cat", "id",
"label"]]
    Numerical_Columns_for_Pipelining = [col for col in
Numerical_Columns_for_Pipelining if col not in ["attack_cat", "id",
"label"]]

    numerical_transformer = Pipeline(steps=[
        ('scaler', PowerTransformer(method='yeo-johnson')),
        ('pca',
PCA(n_components=Numerical_Columns_for_Pipelining.__len__()))
    ])

    categorical_transformer = Pipeline(steps=[
        ('onehot', OneHotEncoder(handle_unknown='ignore',
sparse_output=False))
    ])

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numerical_transformer, Numerical_Columns_for_Pipelining),
            ('cat', categorical_transformer,
Categorical_Columns_for_Pipelining)
        ],
        remainder='passthrough'
    )

    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', model)
    ])

    X_train = train_set_clean.drop('attack_cat', axis=1)
    y_train = train_set_clean['attack_cat']

    X_val = val_set_clean.drop('attack_cat', axis=1)
    y_val = val_set_clean['attack_cat']

    label_encoder = LabelEncoder()
```

```
y_train = label_encoder.fit_transform(y_train)
y_val = label_encoder.transform(y_val)

pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_val)

print("Validation Accuracy:", accuracy_score(y_val, y_pred))
```

Preprocessing yang dilakukan adalah sebagai berikut:

- Untuk kolom numerikal, pertama dilakukan scaler menggunakan metode 'yeo-johnson'. Seharusnya bisa menggunakan 'box-cox' tetapi kami mendapatkan error. Kemudian dilakukan PCA dengan jumlah kolom yang sama dengan jumlah kolom sekarang karena menghasilkan akurasi paling tinggi, lebih tinggi daripada melakukan prediksi tanpa melakukan PCA
- Untuk kolom kategorikal, dilakukan OneHotEncoding untuk mengubah format data menjadi machine-learnable.
- Terakhir untuk variabel target, dilakukan LabelEncoding agar dapat digunakan oleh ID3

Perbandingan Model *From Scratch* dan Library

1. K-Nearest Neighbor (KNN)

Dikarenakan waktu eksekusi yang lama, kami memutuskan untuk melakukan uji coba melakukan 10.000 data pertama.

```
print(f"Model loaded from {filename}")
return self

model = KNN(n_neighbors=5, metric='euclidean')

pipeliner(model)

[62] ✓ 1m 34.8s

... {'dpkts', 'synack', 'spkts', 'ct_dst_src_ltm'}
(7564, 40)
{'dpkts', 'synack', 'spkts', 'ct_dst_src_ltm'}
(1903, 40)
Validation Accuracy: 0.4025223331581713

▶ ✓ from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=71, weights='distance')
# Uncomment kalo mau dirun, ini lama bgt
pipeliner(model)

[63] ✓ 0.5s

... {'dpkts', 'synack', 'spkts', 'ct_dst_src_ltm'}
(7564, 40)
{'dpkts', 'synack', 'spkts', 'ct_dst_src_ltm'}
(1903, 40)
Validation Accuracy: 0.4030478192327903
```


2. Naive Bayes

```
✓ 28s [58] model = NaiveBayes()
        pipeliner(model)

⇌ {'synack', 'spkts', 'ct_dst_src_ltm', 'dloss'}
   (132685, 40)
   {'synack', 'spkts', 'ct_dst_src_ltm', 'dloss'}
   (33178, 40)
<ipython-input-57-f8dc687ced5b>:24: RuntimeWarning: divide by zero encountered in log
likelihood_log = np.sum(np.log(self._gaussian_likelihood(test_row, c)))
Validation Accuracy: 0.10943999035505456

✓ 10s [45] from sklearn.naive_bayes import GaussianNB

        model = GaussianNB()
        pipeliner(model)

⇌ Validation Accuracy: 0.6133883898969197
```

Hasil yang didapatkan model yang sudah dibuat memiliki validation accuracy 0.109, berada di bawah nilai validation accuracy menggunakan model scikit-learn. Hal ini mungkin dapat diatribusikan ke algoritmanya yang banyak melakukan perhitungan matematis. Perhitungan ini memerlukan perhitungan yang presisi yang mungkin terhilangkan pada kode model yang dibuat. Perhitungan yang presisi mungkin dapat dilakukan dengan menggunakan library dan metode yang lebih kompleks.

Selain itu, Naive Bayes merupakan model yang sangat sensitif terhadap dependensi data. Model ini mengasumsikan bahwa semua fitur independen terhadap yang lainnya. Jika terdapat beberapa fitur yang saling berkorelasi dan dependen, hal ini dapat membuat kinerja model menjadi lebih buruk.

3. ID3

```
def load_from_file(self, filename):
    with open(filename, 'r') as f:
        self.tree = json.load(f)
    print(f"Model loaded from {filename}")
    return self

model = ID3(max_depth=5)
pipeliner(model)

[36] ✓ 28m 28.5s

... Validation Accuracy: 0.7067966280295047

from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='entropy')
pipeliner(model)

[37] ✓ 2.9s

... Validation Accuracy: 0.7207586933614331
```

Hasil perbandingan menunjukkan bahwa hanya terdapat 0,014 perbedaan akurasi antara implementasi ID3 dan algoritma dari library. Kita dapat menyimpulkan bahwa algoritma telah diimplementasikan dengan cukup baik, namun *runtime* yang dibutuhkan sangatlah besar.

Pembagian Tugas

NIM	Tugas
13522127	Data Cleaning and Preprocessing
13522138	ID3 Implementation
13522149	K-Nearest Neighbour
13522158	Naive Bayes

Referensi

<https://medium.com/geekculture/step-by-step-decision-tree-id3-algorithm-from-scratch-in-python-no-fancy-library-4822bbfdd88f>