Implementasi Algoritma KNN dan Naive Bayes

Dibuat sebagai tugas besar Mata Kuliah Inteligensi Buatan - IF3170

Dibuat oleh:

Henry Anand Septian Radityo	13521004
Matthew Mahendra	13521007
Hidayatullah Wildan Ghaly Buchary	13521015
Ahmad Nadil	13521024

Program Studi Teknik Informatika
Institut Teknologi Bandung
Bandung

2023

Daftar Isi

Bab 1

Implementasi Algoritma KNN

1.1 Algoritma KNN

Algoritma K-Nearest Neighbors (KNN) adalah algoritma *supervised learning* yang biasanya digunakan untuk melakukan klasifikasi kelas terhadap suatu instans data. Algoritma akan membuat suatu model berdasarkan data latih yang sudah ada. Model yang dibentuk akan digunakan untuk dihitung jarak antar atributnya dengan suatu data tes untuk kemudian dihitung jaraknya yang paling kecil.

Untuk semua instans data latih yang selisihnya dengan data tes sudah dihitung, diurutkan dari yang terkecil hingga terbesar. Dari pengurutan tersebut, diambil k instans data latih untuk dilihat label kelasnya. Label kelas akan dihitung sesuai kategori, e.g. 2 ya 3 tidak 1 ragu. Setelahnya akan diambil label dengan jumlah terbanyak dari instans yang telah diambil. Label tersebut yang akan digunakan sebagai klasifikasinya.

1.2 Implementasi

Implementasi algoritma KNN menggunakan bahasa pemrograman Python dengan pendekatan pemrograman berorientasi objek. Metode fit digunakan untuk mempersiapkan model dan metode predict digunakan untuk melakukan prediksi. Semua input menggunakan tipe dataframe. Perhitungan jarak menggunakan Euclidean Distance dengan perhitungan sebagai berikut,

$$d = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Terdapat konfigurasi untuk proses pelabelan suatu instans. Metode pertama yaitu "uniform" menggunakan metode seperti pada penjelasan algoritma. Metode kedua yaitu "distance" menghitung berdasarkan bobot invers dari jarak suatu data. Dengan perhitungan metode kedua, bisa jadi suatu data yang memiliki jarak yang lebih dekat namun jumlah yang lebih sedikit menjadi yang terpilih karena pembobotan invers dari jarak.

Implementasi algoritma dapat dilihat pada listing 1.1

Listing 1.1: Implementasi Algoritma KNN

```
1 import numpy as np
2 class KNN_Model():
3     # Constructor
4     # Define the number of neighbors to use in prediction
```

```
    # Default neighbors = 5
    def __init__(self, k=3, dist=1, weights="uniform"):
        self.k = k
        self.dist = dist
        self.weights = weights

    # Fit Method
    # Fit the training dataset to the model
    # X are the features, Y is the target
    def fit(self, X, y):
        self.features = X.columns
        self.X_train = X.to_numpy()
        self.y_train = y.to_numpy()

    # Predict Method
    # Prediction using KNN Algorithm
    # X is the data to be predicted
    # Return Predicted Data
    def __predict(self, X):
        if (self.dist == 1):
            distances = [self.__euclidean_distance(X, train) for train
    in self.X_train]
        else:
            # other distance e.g. : Manhattan
            distances = [self.__manhattan_distance(X, train) for train
    in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = None
        if(self.weights == "uniform"):
            most_common = np.bincount(k_nearest_labels)
        elif(self.weights == "distance"):
            weights = 1 / (np.array(distances) + 1e-10)
            chosen_weights = [weights[i] for i in k_indices]
            most_common = np.bincount(k_nearest_labels, weights=
    chosen_weights)
        return np.argmax(most_common)


    def predict(self, X):
        X_ = X.to_numpy()
        result = [self.__predict(test) for test in X_]
        return np.array(result)

    # Distance Method for KNN
    def __euclidean_distance(self,p1,p2):
        p1_numeric = np.array(p1, dtype=float)
        p2_numeric = np.array(p2, dtype=float)
        return np.sqrt(np.sum((p1_numeric - p2_numeric) ** 2))

    def __manhattan_distance(self, p1, p2):
        p1_numeric = np.array(p1, dtype=float)
        p2_numeric = np.array(p2, dtype=float)
        return np.abs(p1_numeric - p2_numeric)
```

Bab 2

Implementasi Algoritma Naive Bayes

2.1 Algoritma Naive Bayes

Naive Bayes adalah algoritma klasifikasi probabilistik yang berdasarkan pada Teorema Bayes. Algoritma ini sangat efektif dalam kasus-kasus di mana dimensi fitur sangat besar, seperti dalam pemrosesan bahasa alami atau analisis teks. Dalam implementasi Naive Bayes yang telah saya buat, setiap fitur dalam dataset dianggap independen dari fitur lainnya, sebuah asumsi yang disebut "naive". Meskipun asumsi ini tampaknya terlalu sederhana atau "naif", dalam banyak kasus, algoritma ini masih menunjukkan performa yang sangat baik.

Algoritma ini bekerja dengan menghitung probabilitas posterior dari setiap kelas berdasarkan input. Ini dilakukan dengan mengalikan probabilitas priori kelas dengan probabilitas dari masing-masing fitur yang diberikan kelas tersebut. Kemudian, kelas dengan probabilitas tertinggi dipilih sebagai hasil prediksi. Probabilitas dihitung menggunakan perhitungan sebagai berikut,

$$P(y|x_1, x_2, \ldots, x_n) = P(y) \times P(x_1|y) \times P(x_2|y) \times \ldots \times P(x_n|y)$$

2.2 Implementasi

Implementasi algoritma Naive Bayes menggunakan bahasa pemrograman Python dengan pendekatan pemrograman berorientasi objek. Metode fit digunakan untuk mempersiapkan model dan metode predict digunakan untuk melakukan prediksi. Metode fit melakukan proses perhitungan $P(y)$ sebagai probabilitas dari target serta menghitung mean dan varian dari fitur. Perhitungan mean dan fitur digunakan karena pendekatan Naive Bayes yang digunakan adalah menggunakan distribusi gauss. Perhitungan $P(x_i|y)$ adalah sebagai berikut,

$$P(x|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(x-\mu_y)^2}{2\sigma_y^2}}$$

Listing 2.1: Implementasi Algoritma Naive Bayes

```
1 class NaiveBayes:
2     def fit(self, X, y):
3         n_samples, n_features = X.shape
```

```
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # calculate mean, var, and prior for each class
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64
    )
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors = np.zeros(n_classes, dtype=np.float64)

        for idx, c in enumerate(self._classes):
            X_c = X[y == c]
            self._mean[idx, :] = X_c.mean(axis=0)
            self._var[idx, :] = X_c.var(axis=0)
            self._priors[idx] = X_c.shape[0] / float(n_samples)


    def predict(self, X):
        y_pred = [self._predict(test) for test in np.array(X)]
        return np.array(y_pred)

    def _predict(self, test):
        posteriors = []

        # calculate posterior probability for each class
        for idx, c in enumerate(self._classes):
            prior = np.log(self._priors[idx])
            posterior = np.sum(np.log(self._pdf(idx, test)))
            posterior = posterior + prior
            posteriors.append(posterior)

        # return class with the highest posterior
        return self._classes[np.argmax(posteriors)]

    def _pdf(self, class_idx, x):
        mean = self._mean[class_idx]
        var = self._var[class_idx]
        numerator = np.exp(-((x - mean) ** 2) / (2 * var + 1e-10))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator
```

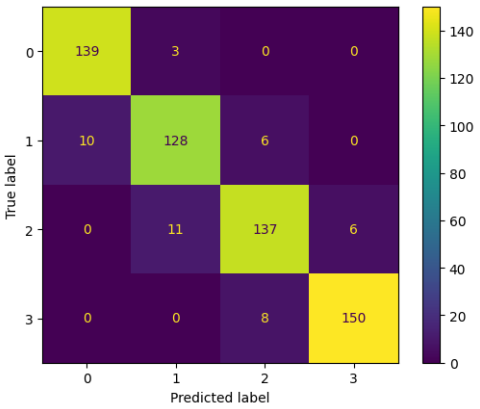
Bab 3

Perbandingan Hasil Prediksi Algoritma dengan Pustaka Scikit-Learn

3.1 Perbandingan KNN

3.1.1 Hasil Prediksi KNN

Hasil prediksi menggunakan KNN mendapatkan skor 0.93 untuk metrk F1, Precision, dan recall. *Preprocessing* yang dilakukan akan dijelaskan pada bab berikutnya. Confusion matrix yang dihasilkan dapat dilihat pada gambar 3.1.



Gambar 3.1: Confusion Matrix untuk Prediksi KNN

3.1.2 Perbandingan dengan Pustaka Scikit

Implementasi pustaka Scikit-learn KNeighborsClassifier menghasilkan hasil yang serupa dengan implementasi algoritma K-Nearest Neighbors (KNN) oleh kelompok kami ketika menggunakan nilai k yang sama. Perbandingan ini dengan cara membandingkan akurasi. Pustaka Scikit-learn KNeighborsClassifier ini menawarkan fleksibilitas dalam pemilihan metode perhitungan jarak, termasuk opsi untuk menggunakan jarak Euclidean, Manhattan, atau Minkowski. Selain itu, Scikit-learn KNeighborsClassifier memungkinkan penggunaan bobot dalam perhitungan, seperti bobot yang seragam dimana semua tetangga

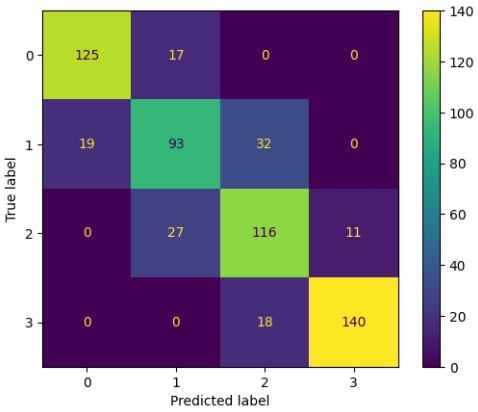
dianggap sama pentingnya, atau bobot berdasarkan jarak, di mana tetangga yang lebih dekat memiliki pengaruh yang lebih besar dalam menentukan klasifikasi. Dalam pengimplementasian Scikit-learn pada KNN, kami memberikan input berupa n_neighbors bernilai k yang sama dengan KNN yang kami buat dan weights bernilai "distance" pada fungsi KNeighborsClassifier. Setelah itu dilakukan fit dengan masukan X_train dan Y_train. Hasil prediksi menggunakan data validasi menghasilkan nilai yang sama dengan prediksi dengan KNN yang dibuat. Karena itu, dapat disimpulkan bahwa algoritma yang dibuat adalah sama dengan algoritma pada pustaka Scikit.

3.2 Scikit Naive Bayes

Implementasi pustaka Scikit-learn GaussianNB menghasilkan hasil yang sama dengan implementasi algoritma Naive Bayes oleh kelompok kami. Perbandingan ini dengan cara membandingkan akurasi. Pustaka GaussianNB di Scikit-learn fokus pada penerapan Naive Bayes untuk fitur yang diasumsikan mengikuti distribusi Gaussian (normal), dan tidak menyediakan metode perhitungan jarak yang dapat dipilih. Berbeda dengan beberapa algoritma lain yang mungkin menawarkan opsi menggunakan jarak secara murni atau menggunakan bobot, GaussianNB bekerja berdasarkan prinsip dasar Naive Bayes yang mengasumsikan independensi antar fitur dan menggunakan probabilitas untuk mengklasifikasikan data. Dalam pengimplementasian Scikit-learn pada Naive Bayes, kami memberikan input berupa input kosong pada fungsi GaussianNB. setelah itu dilakukan fit denagn masukan X_train dan Y_train.

3.2.1 Hasil Prediksi Naive Bayes

Hasil Prediksi Naive Bayes mendapatkan skor akurasi sebesar 0.79 dengan precision dan recall sebesar 0.79 sehingga mendapatkan F1-score sebesar 0.79. *Preprocessing* yang dilakukan akan dijelaskan pada bab berikutnya. Confusion matrix yang dihasilkan dapat dilihat pada gambar 3.2.



Gambar 3.2: Confusion Matrix untuk Prediksi Naive Bayes

3.2.2 Perbandingan dengan Pustaka Sckit

Implementasi pustaka Scikit-learn untuk model Naive bayes menghasilkan akurasi, precision, dan recall yang sama dengan implementasi model dari kelompok kami. Hal ini

membuktikan bahwa implementasi dari naive bayes memberikan hasil yang positif dikarenakan parameter yang diberikan sama dengan default dari pustaka scikit. Data yang dimasukkan untuk kedua model juga merupakan data yang diproses dengan pipeline yang sama sehingga dipastikan hasil prediksi dari model kami dan pustaka merupakan hasil yang sama.

3.3 Insight dari Implementasi Scikit KNN dan Scikit Naive Bayes

Dalam analisis kami, penggunaan Scikit-learn KNeighborsClassifier dan GaussianNB menunjukkan kemampuan yang efektif dalam mengklasifikasikan data, mirip dengan implementasi algoritma yang kami kembangkan sendiri. Khusus untuk KNeighborsClassifier, keunggulannya terletak pada fleksibilitas dalam pemilihan metode perhitungan jarak dan penggunaan bobot. Faktor ini sangat penting karena memungkinkan penyesuaian algoritma sesuai dengan sifat data yang spesifik. Dengan menggunakan bobot berdasarkan jarak, dimana tetangga yang lebih dekat memiliki pengaruh yang lebih besar, kami dapat meningkatkan akurasi klasifikasi. Hal ini menunjukkan bahwa Scikit-learn tidak hanya menyediakan alat yang efisien tetapi juga fleksibel, memungkinkan penyesuaian yang dapat meningkatkan performa model tergantung pada kebutuhan dataset tertentu. Hasil prediksi menggunakan data validasi menghasilkan nilai yang sama dengan prediksi dengan Naive Bayes yang dibuat. Karena itu, dapat disimpulkan bahwa algoritma yang dibuat adalah sama dengan algoritma pada pustaka Scikit.

Implementasi GaussianNB pada Scikit-learn menawarkan pendekatan yang berbeda. Algoritma ini mengasumsikan bahwa fitur mengikuti distribusi Gaussian, yang merupakan asumsi yang kuat namun seringkali efektif dalam berbagai kasus. Tidak adanya kebutuhan untuk pemilihan metode perhitungan jarak atau penggunaan bobot menunjukkan simplicitas dari algoritma Naive Bayes, yang mengandalkan probabilitas bersyarat dan asumsi independensi antar fitur. Hal ini dapat sangat berguna dalam kasus-kasus di mana hubungan antara fitur tidak terlalu rumit atau ketika distribusi data cukup normal. Kemudahan implementasi dan efisiensi komputasi dari GaussianNB menjadikannya pilihan yang menarik untuk banyak masalah klasifikasi, terutama ketika diperlukan solusi yang cepat dan sederhana.

Bab 4

Pemrosesan Data untuk Submisi Kaggle

4.1 Pemrosesan

Pemrosesan yang dilakukan adalah membuat pipeline untuk membantu pembersihan data. Pipeline yang dilakukan adalah sebagai berikut:

1. Menghapus data outlier khususnya di atribut ‘fc‘
2. Mengganti nilai invalid pada semua atribut dengan median
3. Melakukan filter atribut sehingga menyisakan beberapa atribut yaitu battery_power, int_memory, mobile_wt, px_height, px_width, ram, sc_h dan kolom target

Dilakukan penghapusan outlier untuk membantu model dalam melakukan training lalu untuk semua nilai invalid seperti pada kolom sc_w terdapat beberapa nilai 0. Hal ini mustahil apabila dilihat dari metadata yang diberikan. Selanjutnya tahap terakhir dari pipeline adalah menghilangkan semua atribut yang memiliki korelasi yang kurang dan persebaran yang kurang merata sehingga model seperti knn akan lebih baik dalam mencari suatu jarak dari antara kolom test dan kolom train.

Listing 4.1: Pipeline

```
class OutlierRemoval(BaseEstimator, TransformerMixin):
    def __init__(self, columns=None):
        self.columns = columns
        self.lower_bounds = {}
        self.upper_bounds = {}

    def fit(self, X, y=None):
        if self.columns is None:
            self.columns = X.columns

        for column in self.columns:
            q3 = X[column].quantile(q=0.75)
            q1 = X[column].quantile(q=0.25)
            IQR = q3 - q1
            lower_bound = q1 - 1.5 * IQR
            upper_bound = q3 + 1.5 * IQR

            self.lower_bounds[column] = lower_bound
```

```
            self.upper_bounds[column] = upper_bound

        return self

    def transform(self, X):
        for column in self.columns:
            X = X[(X[column] >= self.lower_bounds[column]) & (X[column] <= self.upper_bounds[column])]
        return X

class InvalidReplacement(BaseEstimator, TransformerMixin):
    def __init__(self):
        return None

    def fit(self, X, y=None):
        self.median = X['px_height'].median()
        self.columns = X.columns
        return self

    def transform(self, X):
        X_ = X.copy()
        for col in self.columns:
            if col != 'price_range':
                X_[col] = X_[col].replace(0, X_[col].median())
        return X_

class DropColumn(BaseEstimator, TransformerMixin):
    def __init__(self, column):
        self.column = column

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        X_ = X.drop(columns = self.column)
        return X_

pipeline = Pipeline([
    ('removing outliers', OutlierRemoval(columns=numeric_columns)),
    ('invalid data replacement', InvalidReplacement()),
    ('feature selection', DropColumn(non_numeric_columns + ['fc'] + remove_col))
])
```

Lampiran A

Pembagian Tugas

NIM	Pekerjaan
13521004	KNN, Submisi Kaggle
13521007	KNN, Scikit, Submisi Kaggle
13521015	Naive Bayes, Submisi Kaggle
13521024	Naive Bayes, Submisi Kaggle

Tabel A.1: Pembagian Kerja

Lampiran B

Github dan Notebook

- Repository Github: `https://github.com/MHEN2606/Tubes2_melihatA`
- Kaggle Notebook: `https://www.kaggle.com/code/matthewmahendra/tubes2-melihata/notebook`