Daftar Pustaka

Dhammajoti, D. (2021). *Implementasi ensemble learning menggunakan logistic regression, naive bayes, dan SVM untuk analisis sentimen bahasa Indonesia*. (Tesis Magister, Universitas Multimedia Nusantara). Diakses dari https://kc.umn.ac.id/id/eprint/18017/

Febiawan, M. H., Setiawan, A., & Primadewi, A. (2020). Sistem Pendeteksi Dini Plagiarisme Menggunakan Algoritma Levenshtein Distance. *Jurnal Komtika (Komputasi Dan Informatika)*, *3*(1), 18–27. https://doi.org/10.31603/komtika.v3i1.3464

Jain, V., Kavitha, H., & Mohana Kumar, S. (2022). *Credit Card Fraud Detection Web Application using Streamlit and Machine Learning*. *IEEE International Conference on Data Science and Information System, ICDSIS 2022*, *July*, 1–5. https://doi.org/10.1109/ICDSIS55133.2022.9915901

Kiding, A. (2019). Analisis Sentimen Publik Terhadap Tes Cpns Melalui Media Twiter Menggunakan Metode Naïve Bayes Classifier. *Http://E-Journal.Uajy.Ac.Id/7244/4/3TF03686.Pdf*, *2010*, 15–48.

Kumari, S., Kumar, D., & Mittal, M. (2021). *An ensemble approach for classification and prediction of diabetes mellitus using soft voting classifier*. *International Journal of Cognitive Computing in Engineering*, Januari, 40–46. https://doi.org/10.1016/j.ijcce.2021.01.001

Mahabub, A., Mahmud, M. I. & Hossain, M. F., 2019. *A Robust System for Message Filtering Using an Ensemble Machine Learning Supervised Approach*. *An International Journal of Research and Surveys*, 10(9), pp. 805-811.

Misra, S., Li, H. & He, J., 2020. *Machine Learning for Subsurface Characterization*. Oxford: Gulf Professional Publishing.

Onan, A., Korukoglu, S. & Bulut, H., 2016. *Ensemble of keyword extraction methods and classifiers in text classification*. *Expert System With Applications*, pp. 232-247

Purba, Andry H., & Zakarias Situmorang. Analisis Perbandingan Algoritma Rabin-Karp dan Levenshtein Distance dalam Menghitung Kemiripan Teks. *Jurnal Teknik Informatika Unika Santo Thomas*, vol. 2, no. 2, 2017, pp. 24-32, doi:10.17605/jti.v2i2.187.

Putra, N. P., Rabin-karp, A., Recognition, M. S., Force, B., Karp, R., & Masalah, P. (2019). *292682-Penerapan-Algoritma-Rabin-Karp-Dengan-Pe-73De35F7*. *1*(2), 49–58.

Roakch, L., 2019. *Ensemble Learning Pattern Classification Using Ensemble Method*. 2nd penyunt. Toh Tuck Link: World Scientific Publishing.

Setia, L. D., & Lestariningsih, T. (2018). Perancangan Detektor Plagiarisme Source Code Pada Bahasa Pemrograman Java. *Senatik*, (2018), 55–60.

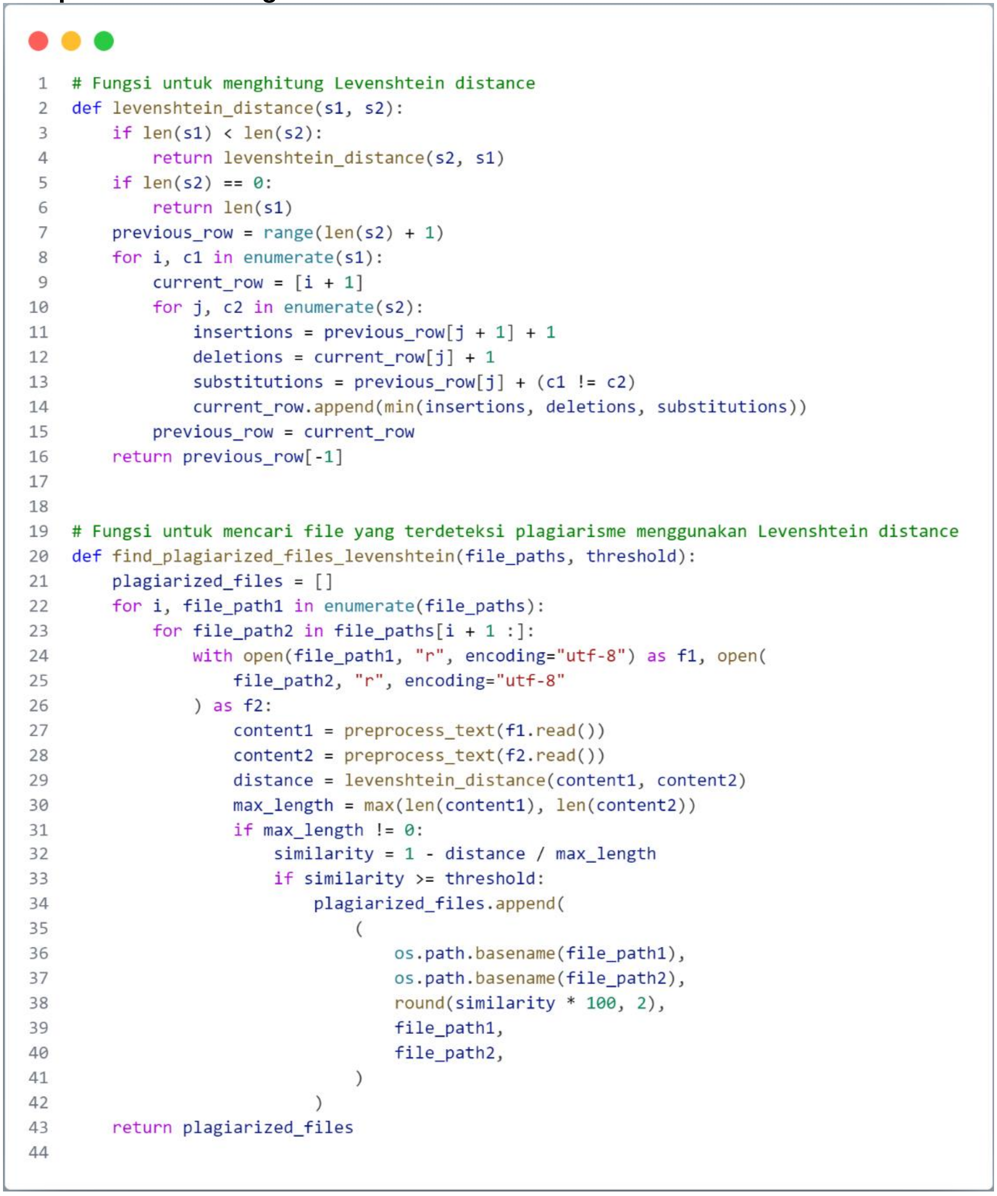
Siahaan, A. P. U., Aryza, S., Hariyanto, E., Rusiadi, Lubis, A. H., Ikhwan, A., & Kan, P. L. E. (201 8). Combination of levenshtein distance and rabin-karp to improve the accuracy of document equivalence level. *International Journal of Engineering and Technology(UAE)*, *7*(2 Special Issue 27), 17–21. https://doi.org/10.14419/ijet.v7i2.27.12084

Turnitin. Similarity score ranges. Turnitin Help Center. Diakses pada 30 Juni 2024, dari https://help.turnitin.com/feedback-studio/turnitin-website/student/the-similarity-report/similarity-score-ranges.htm

Widjaja, T., Gunawan, A., & Liliana, L. (2022). Deteksi Plagiarisme pada Kode Bahasa Pemrograman Java menggunakan XGBoost. *Jurnal Infra*. https://publication.petra.ac.id/index.php/teknik-informatika/article/download/12643/10940

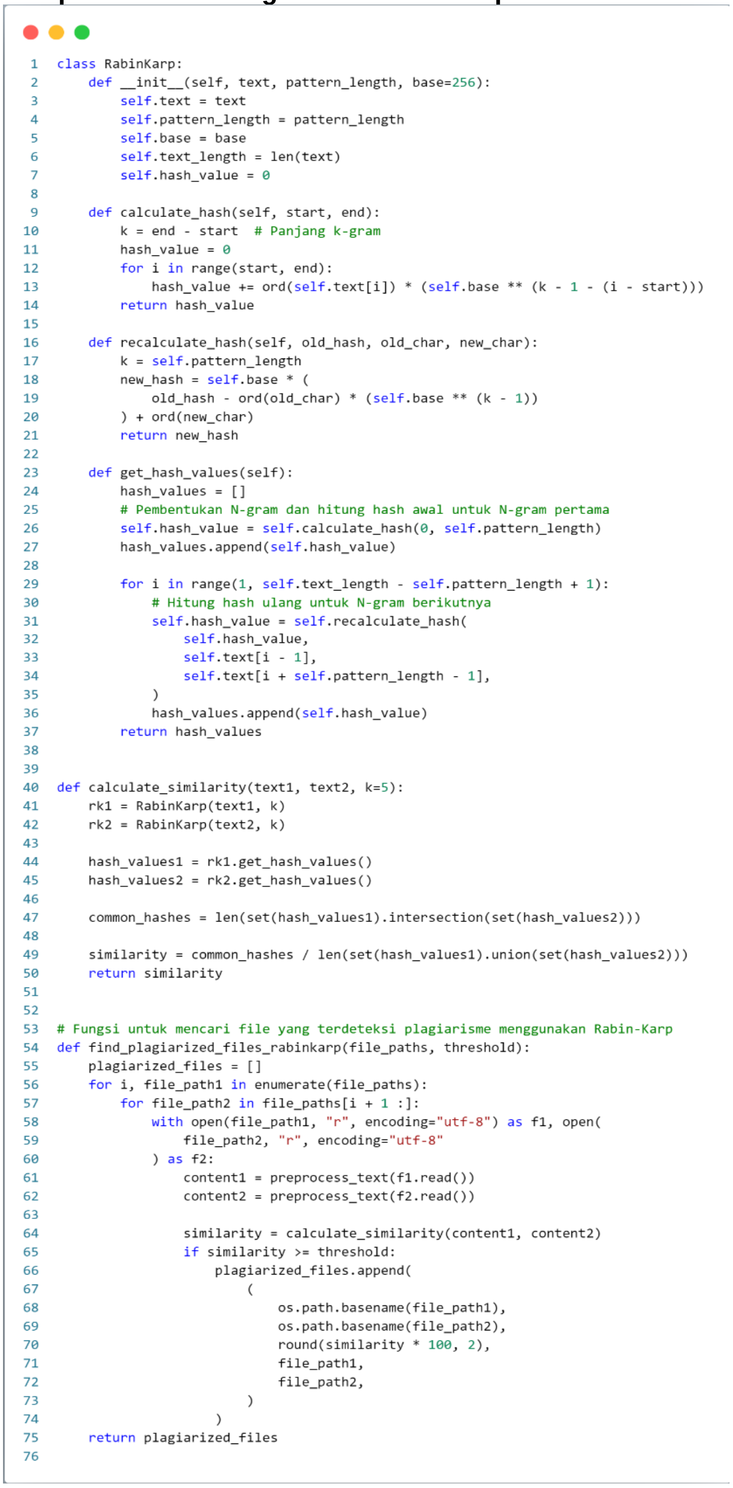
LAMPIRAN

Lampiran 1. Kode Algoritma Levenshtein Distance



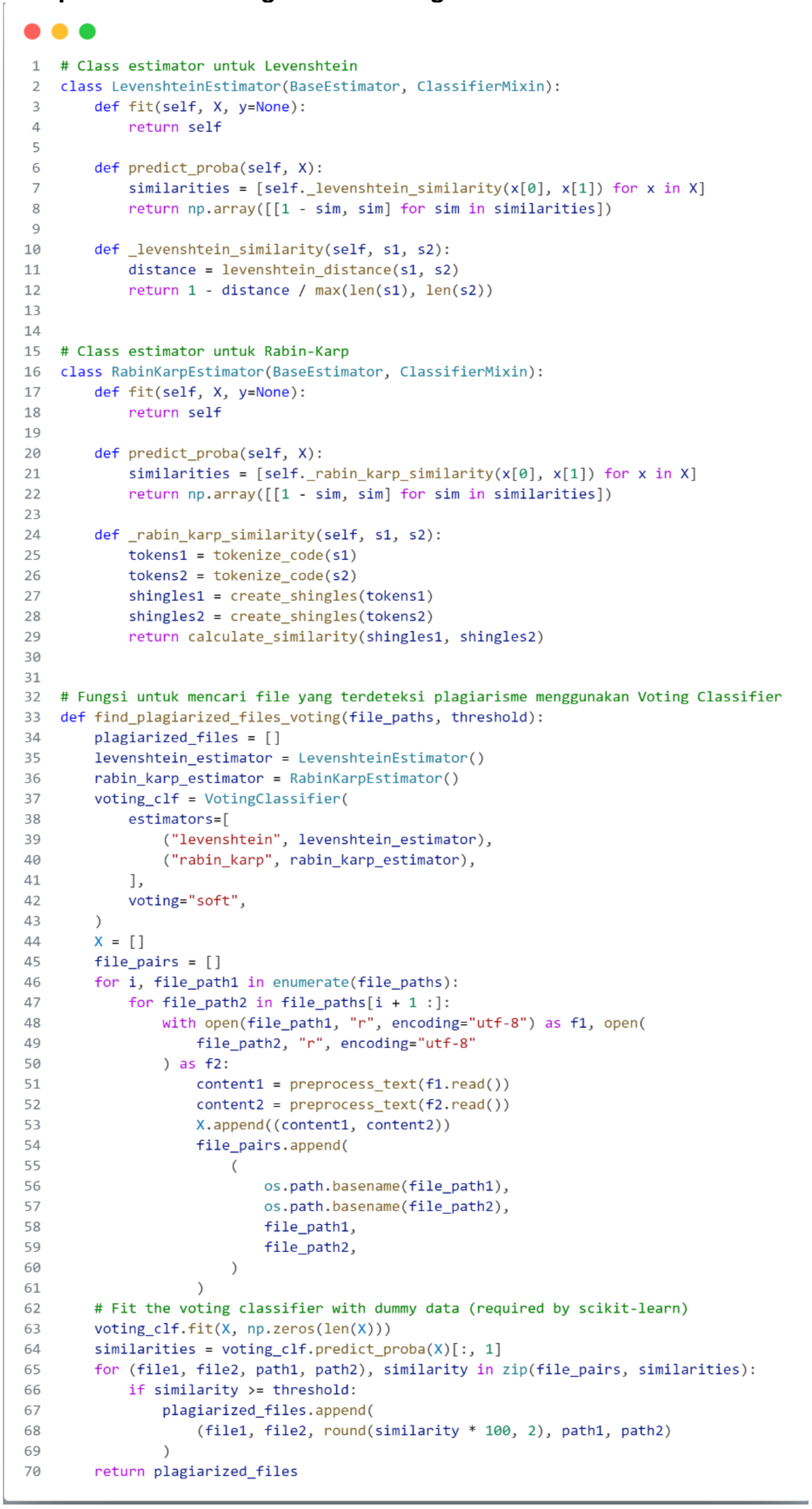
```python
# Fungsi untuk menghitung Levenshtein distance
def levenshtein_distance(s1, s2):
    if len(s1) < len(s2):
        return levenshtein_distance(s2, s1)
    if len(s2) == 0:
        return len(s1)
    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
    return previous_row[-1]


# Fungsi untuk mencari file yang terdeteksi plagiarisme menggunakan Levenshtein distance
def find_plagiarized_files_levenshtein(file_paths, threshold):
    plagiarized_files = []
    for i, file_path1 in enumerate(file_paths):
        for file_path2 in file_paths[i + 1 :]:
            with open(file_path1, "r", encoding="utf-8") as f1, open(
                file_path2, "r", encoding="utf-8"
            ) as f2:
                content1 = preprocess_text(f1.read())
                content2 = preprocess_text(f2.read())
                distance = levenshtein_distance(content1, content2)
                max_length = max(len(content1), len(content2))
                if max_length != 0:
                    similarity = 1 - distance / max_length
                    if similarity >= threshold:
                        plagiarized_files.append(
                            (
                                os.path.basename(file_path1),
                                os.path.basename(file_path2),
                                round(similarity * 100, 2),
                                file_path1,
                                file_path2,
                            )
                        )
    return plagiarized_files

```

Lampiran 2. Kode Algoritma Rabin-Karp



```python
class RabinKarp:
    def __init__(self, text, pattern_length, base=256):
        self.text = text
        self.pattern_length = pattern_length
        self.base = base
        self.text_length = len(text)
        self.hash_value = 0

    def calculate_hash(self, start, end):
        k = end - start  # Panjang k-gram
        hash_value = 0
        for i in range(start, end):
            hash_value += ord(self.text[i]) * (self.base ** (k - 1 - (i - start)))
        return hash_value

    def recalculate_hash(self, old_hash, old_char, new_char):
        k = self.pattern_length
        new_hash = self.base * (
            old_hash - ord(old_char) * (self.base ** (k - 1))
        ) + ord(new_char)
        return new_hash

    def get_hash_values(self):
        hash_values = []
        # Pembentukan N-gram dan hitung hash awal untuk N-gram pertama
        self.hash_value = self.calculate_hash(0, self.pattern_length)
        hash_values.append(self.hash_value)

        for i in range(1, self.text_length - self.pattern_length + 1):
            # Hitung hash ulang untuk N-gram berikutnya
            self.hash_value = self.recalculate_hash(
                self.hash_value,
                self.text[i - 1],
                self.text[i + self.pattern_length - 1],
            )
            hash_values.append(self.hash_value)
        return hash_values


def calculate_similarity(text1, text2, k=5):
    rk1 = RabinKarp(text1, k)
    rk2 = RabinKarp(text2, k)

    hash_values1 = rk1.get_hash_values()
    hash_values2 = rk2.get_hash_values()

    common_hashes = len(set(hash_values1).intersection(set(hash_values2)))

    similarity = common_hashes / len(set(hash_values1).union(set(hash_values2)))
    return similarity


# Fungsi untuk mencari file yang terdeteksi plagiarisme menggunakan Rabin-Karp
def find_plagiarized_files_rabinkarp(file_paths, threshold):
    plagiarized_files = []
    for i, file_path1 in enumerate(file_paths):
        for file_path2 in file_paths[i + 1 :]:
            with open(file_path1, "r", encoding="utf-8") as f1, open(
                file_path2, "r", encoding="utf-8"
            ) as f2:
                content1 = preprocess_text(f1.read())
                content2 = preprocess_text(f2.read())

                similarity = calculate_similarity(content1, content2)
                if similarity >= threshold:
                    plagiarized_files.append(
                        (
                            os.path.basename(file_path1),
                            os.path.basename(file_path2),
                            round(similarity * 100, 2),
                            file_path1,
                            file_path2,
                        )
                    )
    return plagiarized_files
```

Lampiran 3. Kode Algoritma *Voting Classifier*



```python
# Class estimator untuk Levenshtein
class LevenshteinEstimator(BaseEstimator, ClassifierMixin):
    def fit(self, X, y=None):
        return self

    def predict_proba(self, X):
        similarities = [self._levenshtein_similarity(x[0], x[1]) for x in X]
        return np.array([[1 - sim, sim] for sim in similarities])

    def _levenshtein_similarity(self, s1, s2):
        distance = levenshtein_distance(s1, s2)
        return 1 - distance / max(len(s1), len(s2))


# Class estimator untuk Rabin-Karp
class RabinKarpEstimator(BaseEstimator, ClassifierMixin):
    def fit(self, X, y=None):
        return self

    def predict_proba(self, X):
        similarities = [self._rabin_karp_similarity(x[0], x[1]) for x in X]
        return np.array([[1 - sim, sim] for sim in similarities])

    def _rabin_karp_similarity(self, s1, s2):
        tokens1 = tokenize_code(s1)
        tokens2 = tokenize_code(s2)
        shingles1 = create_shingles(tokens1)
        shingles2 = create_shingles(tokens2)
        return calculate_similarity(shingles1, shingles2)


# Fungsi untuk mencari file yang terdeteksi plagiarisme menggunakan Voting Classifier
def find_plagiarized_files_voting(file_paths, threshold):
    plagiarized_files = []
    levenshtein_estimator = LevenshteinEstimator()
    rabin_karp_estimator = RabinKarpEstimator()
    voting_clf = VotingClassifier(
        estimators=[
            ("levenshtein", levenshtein_estimator),
            ("rabin_karp", rabin_karp_estimator),
        ],
        voting="soft",
    )
    X = []
    file_pairs = []
    for i, file_path1 in enumerate(file_paths):
        for file_path2 in file_paths[i + 1 :]:
            with open(file_path1, "r", encoding="utf-8") as f1, open(
                file_path2, "r", encoding="utf-8"
            ) as f2:
                content1 = preprocess_text(f1.read())
                content2 = preprocess_text(f2.read())
                X.append((content1, content2))
                file_pairs.append(
                    (
                        os.path.basename(file_path1),
                        os.path.basename(file_path2),
                        file_path1,
                        file_path2,
                    )
                )
    # Fit the voting classifier with dummy data (required by scikit-learn)
    voting_clf.fit(X, np.zeros(len(X)))
    similarities = voting_clf.predict_proba(X)[:, 1]
    for (file1, file2, path1, path2), similarity in zip(file_pairs, similarities):
        if similarity >= threshold:
            plagiarized_files.append(
                (file1, file2, round(similarity * 100, 2), path1, path2)
            )
    return plagiarized_files
```

Lampiran 4. Riwayat Hidup

Nama	: Awang Mulya Nugrawan
Tempat/ Tanggal Lahir	: Pangkajene Sidrap / 16 September 2002
Jenis Kelaminn	: Laki-Laki
Agama	: Islam
Suku	: Bugis
Alamat	:Griya Al Amin C2 No.7, Kec Biringkanaya, Kel Sudiang Raya,Kota Makassar
No.Hp	: 082191862002
E-mail	: awangmulyanugrawan@gmail.com
Riwayat Pendidikan	:

1. SD Inpres Mannuruki 2
2. SMPN 25 Makassar
3. SMAN 21 Makassar
4. Program Sarjana (S1) Sistem Informasi,
   Departemen Matematika,
   Fakultas Matematika dan Ilmu Pengetahuan Alam,
   Universitas Hasanuddin