# 4. IMPLEMENTASI SISTEM

Bab ini akan membahas implementasi sistem bardasarkan pada analisa dan desain sistem yang telah dibahas pada bab sebelumnya. Implementasi sistem meliputi instalasi dari *library* yang digunakan, preprocessing data, *training* dan *testing* data berserta penjelasan terhadap *source code* yang disediakan. Implementasi program akan dilakukan melalui sebuah aplikasi berbasis web dengan menggunakan Flask sebagai *framework*.

## 4.1. Implementasi dalam pengerjaan model

### 4.1.1. Implementasi perangkat lunak

Pengerjaan model dilakukan dengan bahasa pemrograman Python dengan penggunaan Google Colab, dengan memanfaatkan beberapa *library* yang disediakan oleh python dengan masing-masing *library*memiliki beberapa fungsi dalam proses menghasilkan model. *Library library* yang digunakan adalah sebagai berikut :

Tabel 4.1.

Tabel Penggunaan *Library* dan Masing – Masing Kegunaanya dalam Pengerjaan Model

| no | library | keterangan |
|---|---|---|
| 1 | drive | Pengambilan *dataset* dari file .csv yang disimpan dalam Google Drive |
| 2 | pandas | Pembentukan *data frame* dari *dataset* yang sudah diambil dan juga proses pembersihan dan transformasi data |
| 3 | Sklearn.metrics | Penilaian dari hasil prediksi yang dihasilkan model |
| 4 | Sklearn.utils, Resample | Dilakukan untuk melakukan oversampling guna membantu menghasilkan data yang seimbang dalam penggunaanya baik dalam proses training maupun testing |
| 5 | Sklearn.preprocessing, LabelEncoder | Memberikan sebuah label berupa bilangan *integer* sebagai pengganti dari *class* yang |

| | | |
|---|---|---|
| | | diprediksi dengan masing-masing *integer* mewakili sebuah *class* tertentu |
| 6 | Sklearn.naive_bayes | Penggunaan model Naïve Bayes |
| 7 | Sklearn, svm | Penggunaan model Support Vector Machine |
| 8 | Sklearn.ensemble, RandomForestClassifier | Penggunaan model Random Forest |
| 9 | Sklearn.tree, DecisionTreeClassifier | Penggunaan model Decision Tree |
| 10 | Sklearn.ensemble, BaggingClassifier | Penggunaan ensemble method dalam bentuk Bagging (Bootsrap Aggregating) yang diterapkan dalam tiap model |
| 11 | Sklearn.ensemble, VotingClassifier | Penggunaan ensemble method dalam bentuk Voting yang diterapkan terhadap semua model |
| 12 | matplotlib | Menampilkan grafik untuk visualisasi data dalam menampilkan *confusion matrix* |
| 13 | seaborn | Menampilkan visualisasi statistika dari *confusion matrix* |
| 14 | xgboost | Penggunaan model Extreme Gradient Boosting |
| 15 | joblib | Ekstraksi model untuk diterapkan kedalam aplikasi berbasis web |

Detail dari *Library* yang digunakan akan dibahas pada segmen program

### 4.1.2. Implementasi sistem

#### 4.1.2.1. Data loading

Segmen 4.1. Menghubungkan Google Colab dengan Dataset dari Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Langkah pertama yang dilakukan meliputi penggunaan *library* "drive" dari Google Colab yang berfungsi sebagai koneksi dari program terhadap database yang disimpan pada Google Drive.

Segmen 4.2. Pengambilan Dataset yang Dibaca Menggunakan *Library* Pandas

```
data = pd.read_csv('drive/My Drive/testing kaggle
data/data skripsi/data_skripsi.csv')
data.head(10)
```

Pengambilan data dilakukan dengan membaca dataset di Google Drive pada *path* `'drive/My Drive/testing kaggle data/data skripsi/data_skripsi.csv'` yang kemudian disimpan sebagai sebuah pandas *dataframe*.

4.1.2.2. Preprocessing normalisasi data

Sebelum data memasuki proses *splitting*, data dinormalisasi menggunakan RobustScaler dari SKLearn. Penggunaan normalisasi dengan Robust dikarenakan dapat memproses data dengan outliers dengan baik, serta tidak memiliki range dalam normalisasi data.

Segmen 4.3. Proses Normalisasi Data

```
scaler = RobustScaler()
columns_to_scale = ['umur', 'weight', 'height', 'blood_pressure_top',
'blood_pressure_bot', 'temp', 'respiration']
ndata[columns_to_scale] =
scaler.fit_transform(ndata[columns_to_scale])
```

4.1.2.3. Model evaluation

Sistem kemudian dilanjutkan dengan melakukan import terhadap *library* yang akan digunakan dalam proses pengecekan validasi dari hasil prediski tiap model.

Segmen 4.4. *Function* Pengambilan Nilai *Evaluation Matrix*

```
def get_evaluation_metrics(y_pred):
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred,
average='macro')
    # macro calculates metrics for each label
    recall = recall_score(y_test, y_pred,
average='macro')

    scoring_dict =  {
        'Accuracy' : f'{acc*100:.2f}%',
        'Precision' : f'{prec*100:.2f}%',
        'Recall' : f'{recall*100:.2f}%'
    }
    return scoring_dict
```

Proses dilanjutkan dengan deklarasi dari *function* yang digunakan untuk menghasilkan *evaluation matrix* yang berisikan *accuracy, precision* dan *recall*.

Segmen 4.5. *Function* untuk Visualisasi *Confusion Matrix*

```
def plot_confusion_matrix(model, X_test, y_test):
    y_pred = model.predict(X_test)

    conf_matrix = confusion_matrix(y_test, y_pred)

    labels = model.classes_

    plt.figure(figsize=(10, 8))
    sns.heatmap(conf_matrix, annot=True, fmt='d',
cmap='Blues',
                xticklabels=labels,
yticklabels=labels)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
 plt.show()
```

*Function* yang dikerjakan berikutnya merupakan sebuah *function* yang digunakan untuk memetakan *confusion matrix* yang dihasilkan dari hasil prediksi model sebagai visualisasi hasil prediksi dari data *testing*.

4.1.2.4. Train-Test Splitting

Segmen 4.6. *Train-Test Splitting*

```
label_encoder = LabelEncoder()
data['diagnosa_Label'] =
label_encoder.fit_transform(data['diagnosa'])
X = data.drop(['diagnosa', 'diagnosa_Label',
'patient', 'obat'], axis=1)
y = data['diagnosa_Label']
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.1, stratify=y, random_state=42)
```

Dalam melakukan prediksi dari diagnosis, *dataset* kemudian akan di-*split* kedalam *training* dan *testing* data dengan *ratio* 9 : 1 terhadap *train* dan *test* data masing-masing. *Label encoder* kemudian diterapkan terhadap masing-masing *class* label dikarenakan persyaratan dari prediksi model XGBoost, dimana class yang diterima dalam proses prediksi merupakan tipe data integer. Atribut dari data yang digunakan meliputi karakteristik pasien dan juga gejala yang dialami yang diambil dengan meng-*drop* atribut ‘diagnosa, obat, patient, diagnosa label’ dengan atribut ‘diagnosa label’ digunakan sebagai label. Proses *splitting* juga menggunakan function ‘`stratify=y`‘ dengan hasil testing data yang diambil secara seimbang.

4.1.2.5. Fitting dan Testing model

Sistem kemudian akan memasuki bagian *training* dan *testing* dari masing-masing model yang diawali dengan optimalisasi dari parameter tiap model untuk menghasilkan hasil terbaik. Setelah mendapatkan parameter dengan hasil optimal dari semua model, program kemudian akan melakukan proses *fitting* dan *testing* untuk menampilakn *evaluation dan confusion matrix* dari setiap model.

Proses *hyperparameter tuning* terhadap *base model* dilakukan dengan mendeklarasikan setiap model secara *default* dan kemudian menggunakan *library* GridSearchCV dalam membandingkan hasil dari setiap parameter. Parameter yang digunakan diambil dari pembahasan pada Bab 3 pada tabel parameter *base model* (pada tabel 3.1, 3.2, 3.3, 3.4, 3.5) dan menggunakan baris pertama sebagai parameter prediksi diagnosis.

Segmen 4.7. Deklarasi model dan parameter dari *base model*

```
models = {
    'Naive Bayes': (nb_model, param_grid_nb),
    'SVM': (svm_model, param_grid_svc),
    'Random Forest': (rf_model, param_grid_rf),
    'Decision Tree': (dtc_model, param_grid_dtc),
    'XGBoost': (xgb_model, param_grid_xgb)
}
```

Proses pertama dari *hyperparameter tuning* meliputi deklarasi dari model beserta parameter yang diujikan kedalam *code*.

Segmen 4.8. *Function* untuk melakukan *hyperparameter tuning*

```
def perform_grid_search(model_name, model, param_grid, X_train,
y_train):
    grid_search = GridSearchCV(estimator=model,
param_grid=param_grid, cv=5, n_jobs=-1, scoring='accuracy')
    grid_search.fit(X_train, y_train)
    cv_results_df = pd.DataFrame(grid_search.cv_results_)
    cv_results_df.to_csv(f'{model_name}_grid_search_results.csv',
index=False)
    print(f"Grid Search Results for {model_name}:")
```

```
        print(cv_results_df[['params', 'mean_test_score', 'std_test_score',
    'rank_test_score']])
        print("\n")
        best_estimator = grid_search.best_estimator_
        print(f"Best parameters for {model_name}:
    {grid_search.best_params_}")
        print(f"Best score for {model_name}:
    {grid_search.best_score_}\n")

        return best_estimator
```

Proses dilanjutkan dengan mendeklarasikan fungsi untuk melakukan proses *fitting* terhadap model yang akan diujicobakan. Fungsi juga meliputi penyimpanan hasil dari masing-masing model kedalam sebuah *file* {nama model}.csv . nilai dari pengujian masing – masing parameter juga ditampilkan dengan parameter terbaik disimpan untuk dideklarasikan sebagai model akhir.

Segmen 4.9. Deklarasi model dan parameter dari *base model*

```
for model_name, (model, param_grid) in models.items():
        print(f"Performing Grid Search for {model_name}...")
        best_estimator = perform_grid_search(model_name, model,
    param_grid, X_train, y_train)
        best_models[model_name] = best_estimator
```

Proses dilanjutkan dengan *looping* yang dilakukan terhadap semua model dan parameter yang telah dideklarasikan untuk melakukan proses *hyperparameter tuning* dengan menggunakan GridSearch. Proses *looping* menghasilkan penyimpanan semua parameter terbaik yang telah digunakan oleh masing-masing model.

Segmen 4.10. Deklarasi ulang model dengan parameter terbaik

```
nb_model = best_models['Naive Bayes']
svm_model = best_models['SVM']
rf_model = best_models['Random Forest']
dtc_model = best_models['Decision Tree']
```

```
xgb_model = best_models['XGBoost']
```

Proses terakhir dari *tuning* meliputi deklarasi ulang terhadap semua model menggunakan parameter terbaik. Proses deklarasi ulang ini digunakan untuk mempersingkat proses deklarasi dari Voting Method dan juga agar masing-masing model dapat dievaluasi durasi proses *fitting* dan juga *evaluation* dan *confusion matrix*-nya.

4.1.2.5.1. Naïve Bayes

Segmen 4.11. *Fitting Base* Model Naïve Bayes

```
nb_model.fit(X_train, y_train)
nb_model.score(X_test, y_test)

joblib.dump(nb_model, 'nb_model.pkl')
```

4.1.2.5.2. Support Vector Machine

Segmen 4.12. *Fitting Base* Model Support Vector Machine

```
svm_model.fit(X_train, y_train)
svm_model.score(X_test, y_test)

joblib.dump(svm_model, 'svm_model.pkl')
```

4.1.2.5.3. Random Forest

Segmen 4.13. *Train-Test Splitting*

```
rf_model.fit(X_train, y_train)
rf_model.score(X_test, y_test)

joblib.dump(rf_model, 'rf_model.pkl')
```

4.1.2.5.4. Decision Tree

Segmen 4.14. *Fitting Base* Model Decision Tree

```
dtc_model.fit(X_train, y_train)
dtc_model.score(X_test, y_test)

joblib.dump(dtc_model, 'dtc_model.pkl')
```

4.1.2.5.5. Extreme Gradient Boosting

Segmen 4.15. *Fitting Base* Model Extreme Gradient Boosting

```
xgb_model.fit(X_train, y_train)
xgb_model.score(X_test, y_test)

joblib.dump(xgb_model, 'xgb_model.pkl')
```

4.1.2.6. Bagging method

Penerapan Bagging Method dilakukan dengan meng-*import* 'BaggingClassifier' dari *library* SK-Learn. Penerapan dari Bagging method dilakukan terhadap masing - masing model yang telah melalui proses *tuning*. Penerapan Bagging Method sendiri juga melalui proses *tuning parameter* terhadap parameter yang telah dibahas pada Bab 3 pada tabel parameter Bagging Method (tabel 3.6) baris pertama sebagai parameter prediksi diagnosis.

Segmen 4.16. *Function* penerapan *hyperparameter tuning* pada Bagging Method

```
def perform_bagging_grid_search(model_name, base_classifier, param_grid, X_train, y_train):
    bagging_model = BaggingClassifier(base_estimator=base_classifier, random_state=42)
    grid_search = GridSearchCV(estimator=bagging_model, param_grid=param_grid, cv=5, n_jobs=-1, scoring='accuracy')
    grid_search.fit(X_train, y_train)
    cv_results_df = pd.DataFrame(grid_search.cv_results_)
    cv_results_df.to_csv(f'{model_name}_bagging_grid_search_results.csv', index=False)
    print(f"Grid Search Results for {model_name} with Bagging Classifier:")
    print(cv_results_df[['params', 'mean_test_score', 'std_test_score', 'rank_test_score']])
    print("\n")
    best_params = grid_search.best_params_
    print(f"Best parameters for {model_name} with Bagging Classifier: {best_params}")
    print(f"Best score for {model_name} with Bagging Classifier: {grid_search.best_score_}\n")
```

```
        return best_params
```

Sama seperti proses *tuning* pada *base model*, proses *tuning* pada Bagging Method meliputi sebuah *function* untuk melakukan proses uji coba terhadap masing – masing model dan juga parameternya. Penggunaan 'random_state' digunakan agar hasil yang diujikan dapat dibandingkan terhadap masing – masing model dengan menggunakan basis yang sama.

Segmen 4.17. Proses *looping* dari *tuning* Bagging Method

```
for model_name, base_classifier in base_classifiers.items():
        print(f"Performing Grid Search for {model_name} with Bagging
    Classifier...")
        best_params[model_name] =
    perform_bagging_grid_search(model_name, base_classifier,
    param_grids[model_name], X_train, y_train)
```

Proses kemudian dilanjutkan dengan melakukan proses *looping* untuk melakukan uji coba terhadap masing – masing model dan parameter yang telah ditetapkan.

Segmen 4.18. Deklarasi ulang dari model dengan Bagging Method

```
        bagging_dtc_classifier = BaggingClassifier(base_estimator=dtc_model,
        **best_params['Decision Tree'])
        bagging_rf_classifier = BaggingClassifier(base_estimator=rf_model,
        **best_params['Random Forest'])
        bagging_nb_classifier = BaggingClassifier(base_estimator=nb_model,
        **best_params['Naive Bayes'])
        bagging_svm_classifier = BaggingClassifier(base_estimator=svm_model,
        **best_params['SVM'])
        bagging_xgb_classifier = BaggingClassifier(base_estimator=xgb_model,
        **best_params['XGBoost'])
```

Dengan akhir proses merupakan deklarasi ulang terhadap semua Bagging Model untuk mempermudah proses deklarasi Voting Method dan penerapan visualisasi *evaluation* dan *confusion matrix*.

4.1.2.6.1. Naïve Bayes

Segmen 4.19. *Fitting* Bagging Method pada Model Naïve Bayes

```
nb_classifier.fit(X_train, y_train)
nb_classifier.score(X_test, y_test)

joblib.dump(nb_classifier, 'nb_classifier.pkl')
```

4.1.2.6.2. Support Vector Machine

Segmen 4.20. *Fitting* Bagging Method pada Model Support Vector Machine

```
bagging_svm_classifier.fit(X_train, y_train)
bagging_svm_classifier.score(X_test, y_test)

joblib.dump(bagging_svm_classifier,
'bagging_svm_classifier.pkl')
```

4.1.2.6.3. Random Forest

Segmen 4.21. *Fitting* Bagging Method pada Model Random Forest

```
bagging_rf_classifier.fit(X_train, y_train)
bagging_rf_classifier.score(X_test, y_test)

joblib.dump(bagging_rf_classifier,
'bagging_rf_classifier.pkl')
```

4.1.2.6.4. Decision Tree

Segmen 4.22. *Fitting* Bagging Method pada Model Decision Tree

```
bagging_dtc_classifier.fit(X_train, y_train)
bagging_dtc_classifier.score(X_test, y_test)

joblib.dump(bagging_dtc_classifier,
'bagging_dtc_classifier.pkl')
```

4.1.2.6.5. Extreme Gradient Boosting

Segmen 4.23. *Fitting* Bagging Method pada Model Extreme Gradient Boosting

```
bagging_xgb_classifier.fit(X_train, y_train)
bagging_xgb_classifier.score(X_test, y_test)

joblib.dump(bagging_xgb_classifier,
'bagging_xgb_classifier.pkl')
```

4.1.2.7. Voting method

Penggunaan Voting Method diawali dengan melakukan proses *import* 'VotingClassifier' dari *library* SK-Learn. Voting Method dilakukan dengan menggabungkan beberapa *estimator* kedalam sebuah *classifier* yang akan menghasilkan *output* berupa *class label* dari pengambilan suara mayoritas dari *classifier* tersebut. Dalam penelitian skripsi ini, Voting Method dilakukan terhadap kelima model dasar dan kelima model dengan bagging method, dimana masing-masing telah melalui proses *hyperparameter tuning*. Model dari Voting Method sendiri kemudian akan melalui proses *tuning* sesuai dengan yang dibahas pada Bab 3 pada tabel 3.7.

Segmen 4.24. *Function* untuk melakukan *hyperparameter tuning* pada Voting Method

```
def perform_voting_grid_search(voting_classifier, model_name, X_train,
y_train):
        grid_search = GridSearchCV(estimator=voting_classifier,
    param_grid=param_grid, cv=5, n_jobs=-1, scoring='accuracy')
        grid_search.fit(X_train, y_train)
        cv_results_df = pd.DataFrame(grid_search.cv_results_)
        cv_results_df.to_csv(f'{model_name}_grid_search_results.csv',
    index=False)
        print(f"Grid Search Results for {model_name}:")
        print(cv_results_df[['params', 'mean_test_score', 'std_test_score',
    'rank_test_score']])
        print("\n")
        best_estimator = grid_search.best_estimator_
        print(f"Best parameters for {model_name}:
    {grid_search.best_params_}")
        print(f"Best score for {model_name}: {grid_search.best_score_}\n")
        return best_estimator
```

Proses *tuning* dari voting method dimulai dengan mendeklarasikan fungsi untuk melakukan pengujian terhadap model dan parameter dengan menggunakan GridSearch.

Segmen 4.25. Proses pengujian dari Voting Method dengan parameter yang diujikan

```
print("Performing Grid Search for Voting Classifier (Base Models)...")
best_voting_classifier_base =
perform_voting_grid_search(voting_classifier_base,
"Voting_Classifier_Base", X_train, y_train)

print("Performing Grid Search for Voting Classifier (Bagged Models)...")
best_voting_classifier_bagged =
perform_voting_grid_search(voting_classifier_bagged,
"Voting_Classifier_Bagged", X_train, y_train)
```

Proses dilanjutkan dengan penerapan fungsi yang sudah dideklarasikan kedalam model yang berisi *voting* terhadap *base model* dan model yang menggunakan Bagging Method dan parameter yang diujikan.

Segmen 4.26. Deklarasi ulang Voting Method dengan parameter terbaik

```
voting_classifier_base_best = VotingClassifier(
  estimators=[
    ('rf', rf_model),
    ('dt', dtc_model),
    ('svm', svm_model),
    ('xgb', xgb_model),
    ('nb', nb_model)
  ],
  voting=best_voting_classifier_base.voting,
  weights=best_voting_classifier_base.weights
)

voting_classifier_bagged_best = VotingClassifier(
  estimators=[
    ('rf', bagging_rf_classifier),
    ('dt', bagging_dtc_classifier),
    ('svm', bagging_svm_classifier),
```

```
            ('xgb', bagging_xgb_classifier),
            ('nb', bagging_nb_classifier)
        ],
        voting=best_voting_classifier_bagged.voting,
        weights=best_voting_classifier_bagged.weights
    )
```

Proses *tuning* diakhiri dengan deklarasi ulang dari model agar dapat dilakukan visualisasi dari *confusion* dan *evaluation matrix*.

4.1.2.7.1. Voting Method terhadap *base model*

Segmen 4.27. *Fitting* Voting Method pada *Base Model*

```
voting_classifier.fit(X_train, y_train)
voting_classifier.score(X_test, y_test)

joblib.dump(voting_classifier, 'voting_model.pkl')
```

Voting Method yang dilakukan terhadap model dasar.

4.1.2.7.2. Voting Method terhadap *bagging model*

Segmen 4.28. *Fitting* Voting Method pada Model dengan Bagging Classifier

```
voting_classifier.fit(X_train, y_train)
voting_classifier.score(X_test, y_test)

joblib.dump(voting_classifier,
'voting_classifier.pkl')
```

Voting Method yang dilakukan terhadap model dari Bagging Classifier

4.1.2.8. Data cleaning untuk prediksi obat

Segmen 4.29. *Exploding* Data Berdasarkan Kolom Obat

```
data['obat'] = data['obat'].str.split(',')
df_exploded = data.explode('obat')
```

Dalam proses prediksi obat, perlu melakukan proses *cleaning* karena tipe dari data obat disimpan sebagai satu *string* yang mengandung beberapa obat. Sebagai contoh, berikut isi dari salah satu atribut obat “

Segmen 4.30. *Cleaning* Data Obat

```
df_exploded['obat'] =
df_exploded['obat'].str.lower().str.strip()
df_exploded['obat'] =
df_exploded['obat'].str.split().str[0]
```

Data obat yang telah dipisah kedalam baris yang berbeda-beda kemudian di cleaning dengan menghapus string

Segmen 4.31. Penggantian Label Obat Sesuai dengan Data Wawancara

```
df_exploded['obat'] =
df_exploded['obat'].replace(replacements)
label_counts = df_exploded['obat'].value_counts()
print(label_counts)
```

Data label obat kemudian diganti sesuai dengan 'replacement' yang disesuaikan dengan hasil penjelasan dari dr. Rita Wey untuk mengurangi waktu dalam pembelajaran yang tidak diperlukan (prediksi obat berbeda dengan efek yang sama).

4.1.2.9. Oversampling data obat

Segmen 4.32. *Oversampling* Data

```
df_exploded = pd.concat([
    resample(df_exploded[df_exploded['obat'] ==
label],
             replace=True,
             n_samples=352,
             random_state=42)
    for label, n_samples in
df_exploded['obat'].value_counts().items()
])
```

Dataset obat kemudian di transformasi dengan cara melakukan oversampling terhadap tiap obat sebagai *class* sebanyak 352 data per *class label* dengan total data 35.200 data.

4.1.2.10. Train-Test Splitting

Segmen 4.33. *Train-Test Splitting*

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import joblib

label_encoder = LabelEncoder()
df_exploded['obat_Label'] =
label_encoder.fit_transform(df_exploded['obat'])
X = df_exploded.drop(['diagnosa', 'obat_Label',
'patient', 'obat'], axis=1)
y = df_exploded['obat_Label']
X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.1, stratify=y, random_state=42)
```

Implementasi sistem dilanjutkan dengan *splitting* data kedalam *training* dan *testing* data sebagai data yang akan digunakan untuk melatih dan mengevaluasi model.

4.1.2.11. Testing model

Proses testing dari dataset obat terhadap setiap model dasar menggunakan proses *fitting* yang sama dengan pada prediksi diagnosis, tetapi menggunakan parameter yang berbeda sesuai yang dideklarasikan pada Bab 3 pada tabel (pada tabel 3.1, 3.2, 3.3, 3.4, 3.5) pada baris kedua, parameter untuk prediksi obat.

4.1.2.12. Bagging method

Proses *fitting* terhadap masing-masing model dengan menggunakan metode Bagging memiliki perbedaan dalam parameter yang diujikan. Hal ini dikarenakan keterbatasan dalam waktu dan *hardware* dimana semakin tinggi nilai parameter, semakin lama proses testing terhadap sebuah model dikerjakan. Sesuai dengan pembahasan pada Bab 3.

4.1.2.13. Voting method

Proses *fitting* dari Voting Method terhadap *base model* dan *base model* yang menerapkan Bagging method dengan proses *hyperparameter tuning* yang sama dengan prediksi diagnosis, dengan pengujian parameter diterapkan terhadap parameter 'weights' dalam pemberian bobot terhadap masing-masing model.

4.2. Implementasi dalam pengerjaan aplikasi berbasis web untuk validasi

4.2.1. Implementasi perangkat lunak

Pengerjaan aplikasi berbasis web menggunakan bahasa pemrograman Python sebagai *back-end* dengan menerapkan *framework* Flask sebagai penghubung dengan Bahasa pemrograman HTML sebagai *front-end*. Dalam pengerjaanya, aplikasi berbasis web juga menggunakan beberapa *library* dari Python.

4.2.2. Implementasi sistem

4.2.2.1. Implementasi dari *back-end* program

Segmen 4.34. *Framework* dari *Library* Flask

```
app = Flask(__name__)
if __name__ == '__main__':
app.run(debug=True)
```

Proses dasar pembuatan *framework*

Segmen 4.35. *Function convert value*

```
def convert_checkbox_value(value):
```

```
    return 1 if value == 'on' else 0
```

*Funtion* 'convert_checkbox_value' digunakan dalam pengambilan data dari *page* informasi pasien yang berguna untuk mengubah tipe data dari data gejala yang dialami pasien, dimana data yang awalnya berbentuk 'on' bila dicentang kemudian dikonversikan menjadi sebuah data biner yang berisi 1 bila 'on' dan 0 bila bukan.

Segmen 4.36. *Label Mapping* untuk Proses *Decoding* Hasil Prediksi

```
class_mapping = {'j06.9': 3, 'i10': 2, 'm79.1': 7, 'r50.9': 8,
'k30': 5, 'l23.9': 6, 'a09': 0, 'r51': 9, 'e11': 1, 'j45': 4}
medicine_mapping = {'neurotropik': 66, 'candesartan': 13,
'triamcinolone': 87, 'flunarizine': 36, 'domperidone': 31,
'paracetamol': 72, 'omeprazole': 69, 'amlodipine': 4,
'triprolidine': 89, 'probiotik': 75, 'sukralfat': 85,
'ambroxol': 3, 'metformin': 60, 'meloxicam': 58, 'erdosteine':
33, 'ondansetron': 70, 'orinox': 71, 'glimipiride': 41,
'azithromycin': 7, 'deslotine': 25, 'salbutamol': 81,
'amoxicillin': 5, 'ibuprofen': 44, 'attapulgite': 6,
'tropineuron': 92, 'cefixime': 17, 'kalmeco': 50, 'inhipraz':
46, 'cetirizine': 19, 'benoson': 10, 'diagit': 29, 'lodia':
54, 'simvastatin': 83, 'berotec': 11, 'dexamethasone': 27,
'puyer': 79, 'zinc': 98, 'mirasic': 62, 'terbutaline': 86,
'cefabiotik': 15, 'cefadroxil': 16, 'epsonal': 32,
'demacolin': 24, 'kalium diklofenak': 49, 'lycoxy': 57,
'dexanta': 28, 'mozuku': 63, 'ciprofloxacin': 20, 'neuroxon':
67, 'methylprednisolone': 61, 'insulin': 48, 'loratadine': 56,
'varoc': 94, 'clopidogrel': 22, 'colergis': 23, 'inj': 47,
'flacicox': 35, 'imboost': 45, 'zoline': 99, 'betametason':
12, 'desolex': 26, 'trolac': 90, 'glibenclamide': 40, 'proza':
78, 'allopurinol': 0, 'clinex': 21, 'lodine': 55, 'natrium
diklofenak': 64, 'proceles': 76, 'spironolactone': 84,
'gliquidone': 42, 'xytrol': 96, 'cendo': 18, 'gabapentin': 38,
'pycostein': 80, 'gentamicin': 39, 'ziberid': 97, 'potaflam':
73, 'graphalax': 43, 'metamizole': 59, 'nurcla': 68,
'dianicol': 30, 'aloclair': 1, 'kalmico': 51, 'trombophobe':
91, 'triamnicolon': 88, 'furosemide': 37, 'salgestam': 82,
'levofloxacin': 53, 'cecyl': 14, 'potflam': 74, 'alprazolam':
2, 'vefacef': 95, 'uresix': 93, 'neorovit': 65, 'pronalges':
77, becomzer': 9, 'ketoconazole': 52, 'b1': 8, 'festaric':
34}

reverse_class_mapping = {v: k for k, v in
class_mapping.items()}
```

```
reverse_medicine_mapping = {v: k for k, v in
medicine_mapping.items()}
```

Bagian berikutnya dilanjutkan dengan adanya proses pengembalikan dari proses *label encoding* dimana output prediksi yang awalnya berupa bilangan integer yang mewakilkan *class label* akan di-*decode* menjadi *class label* awal.

Segmen 4.37. *Loading* Model Kedalam Aplikasi Berbasis Web

```
diagnosis_models = [
    joblib.load('dtc_model.pkl'),
    joblib.load('bagging_dtc_classifier.pkl'),
    joblib.load('rf_model.pkl'),
    joblib.load('bagging_xgb_classifier.pkl'),
    joblib.load('voting_model.pkl')
]
medicine_models = [
    joblib.load('dtc_model (1).pkl'),
    joblib.load('bagging_dtc_classifier (1).pkl'),
    joblib.load('rf_model (1).pkl'),
    joblib.load('bagging_rf_classifier (1).pkl'),
    joblib.load('voting_classifier (1).pkl')
]
```

Program kemudian akan melakukan *loading* terhadap model yang telah menghasilkan nilai terbaik berdasarkan nilai *evaluation matrix* dengan masing – masing prediksi menggunakan model yang berbeda, berdasarkan pada proses *data fitting* dari tiap model.

Segmen 4.38. *Route* dalam Flask *Framework* Menuju *Home Page*

```
@app.route('/')
def index():
    # Sample data for different machine learning algorithms
    metrics_data = {
        "algorithms": ["Naive Bayes", "Support Vector
Machine", "Random Forest", "Decision Tree", "Extreme Gradient
Boosting"],
        "accuracy": [0.8892, 0.9489, 0.9602, 0.9659, 0.9602],
        "precision": [0.8822, 0.9484, 0.9594, 0.9654, 0.9594],
        "recall": [0.8889, 0.9488, 0.9602, 0.9659, 0.9602]
    }

    # Sample data for bagging method
    bagging_data = {
```

```
        "algorithms": ["Naive Bayes", "Support Vector
Machine", "Random Forest", "Decision Tree", "Extreme Gradient
Boosting"],
        "bagging_accuracy": [0.9034, 0.9489, 0.9602, 0.9659,
0.9659],
        "bagging_precision": [0.9030, 0.9484, 0.9603, 0.9654,
0.9654],
        "bagging_recall": [0.9032, 0.9488, 0.9602, 0.9659,
0.9659]
    }

    # Sample data for voting algorithm
    voting_data = {
        "algorithms": ["Model Voting", "Bagging Classifier
Voting"],
        "voting_accuracy": [0.9659, 0.9602],
        "voting_precision": [0.9654, 0.9603],
        "voting_recall": [0.9659, 0.9602]
    }
    meds_metrics_data = {
        "algorithms": ["Naive Bayes", "Support Vector
Machine", "Random Forest", "Decision Tree", "Extreme Gradient
Boosting"],
        "accuracy": [0.5026, 0.6636, 0.6722, 0.6733, 0.4074],
        "precision": [0.4502, 0.6188, 0.6196, 0.6274, 0.2728],
        "recall": [0.5022, 0.6634, 0.6721, 0.6731, 0.4074]
    }

    # Sample data for bagging method
    meds_bagging_data = {
        "algorithms": ["Naive Bayes", "Support Vector
Machine", "Random Forest", "Decision Tree", "Extreme Gradient
Boosting"],
        "bagging_accuracy": [0.5037, 0.6659, 0.6733, 0.6750,
0.5545],
        "bagging_precision": [0.4466, 0.6176, 0.6235, 0.6319,
0.4884],
        "bagging_recall": [0.5031, 0.6658, 0.6730, 0.6746,
0.5547]
    }

    # Sample data for voting algorithm
    meds_voting_data = {
        "algorithms": ["Model Voting", "Bagging Classifier
Voting"],
        "voting_accuracy": [0.6702, 0.6753],
        "voting_precision": [0.6235, 0.6313],
```

```
        "voting_recall": [0.6699, 0.6749]
    }
    return render_template('index.html',
                           metrics_data=metrics_data,
                           bagging_data=bagging_data,
                           voting_data=voting_data,
                           meds_metrics_data=meds_metrics_data
,
                           meds_bagging_data=meds_bagging_data
,
                           meds_voting_data=meds_voting_data)
```

Pada proses loading index.html yang akan menjadi *home page* atau halaman pertama dari aplikasi, program akan meneruskan data yang didapat dari proses *scoring* pada Google Colab menuju halaman index.html untuk divisualisasikan ke dalam tabel yang mewakili masing-masing model.

Segmen 4.39. *Route* dalam Flask *Framework* Menuju *Patient Info Page*

```
@app.route('/patient_info')
def patient_info():
    return render_template('patient_info.html')
```

Menjalankan *page* 'patient_info.html' dimana data karakteristik dan juga gejala pasien akan di-*input*-kan untuk diproses oleh model *Machine Learning* sehingga dapat menghasilkan *class label* dari hasil prediksi.

Proses pengambilan data dari *page* patient_info.html dengan menggunakan 'request' dari library Flask.

Proses dilanjutkan dengan melakukan konversi data yang awalnya bertipe String dari *request from* menjadi tipe data bilangan agar dapat diproses oleh model *machine learning*.

Segmen 4.40. Proses Prediksi Diagnosis

```
    diagnosis_predictions = []
    diagnosis_probabilities = []

   for model in diagnosis_models:
        diagnosis_prediction = model.predict(patient_df)[0]
        decoded_diagnosis_prediction = [key for key, value in
class_mapping.items() if value == diagnosis_prediction][0]
        diagnosis_predictions.append(decoded_diagnosis_predict
ion)

        if hasattr(model, 'predict_proba'):
```

```
            probs = model.predict_proba(patient_df)[0]
            print("this is probs", probs)
            diagnosis_probabilities.append(probs[diagnosis_pre
diction] * 100)
        else:
            # Simulate probability estimate if model does not
support predict_proba
            votes = np.array([model.predict(patient_df)[0] for
_ in range(100)])
            class_votes = np.bincount(votes)
            predicted_class_votes =
class_votes[diagnosis_prediction]
            total_votes = len(votes)
            diagnosis_probabilities.append((predicted_class_vo
tes / total_votes) * 100)
```

Proses dilanjutkan dengan melakukan prediksi terhadap data yang diambil dari patient_info.html yang dilanjutkan dengan menghitung probabilitas dari hasil prediksi. Program kemudian melanjutkan dengan mengambil probabilitas dari hasil prediksi dan menyimpanya kedalam *list* probabilitas yang akan diteruskan kepada *page* hasil. Bila model memiliki attribut untuk menghasilkan probabilitas langsung, maka nilai probabilitas akan dihitung berdasarkan persentase prediksi yang dilakukan sebanyak 100 kali.

Segmen 4.41. Pengambilan Diagnosis Kedalam Data Prediksi Obat

```
majority_diagnosis_prediction =
Counter(diagnosis_predictions).most_common(1)[0][0]
    majority_diagnosis_label =
class_mapping[majority_diagnosis_prediction]
    print(majority_diagnosis_label)
```

Hasil diagnosis label yang dihasilkan kemudian akan diambil nilai mayoritasnya untuk menghasilkan hasil terbanyak yang kebudian ditambahkan kedalam data pasien sebagai atribut ‘diagnosa_Label’ untuk prediksi obat.

Segmen 4.42. Proses Prediksi Obat

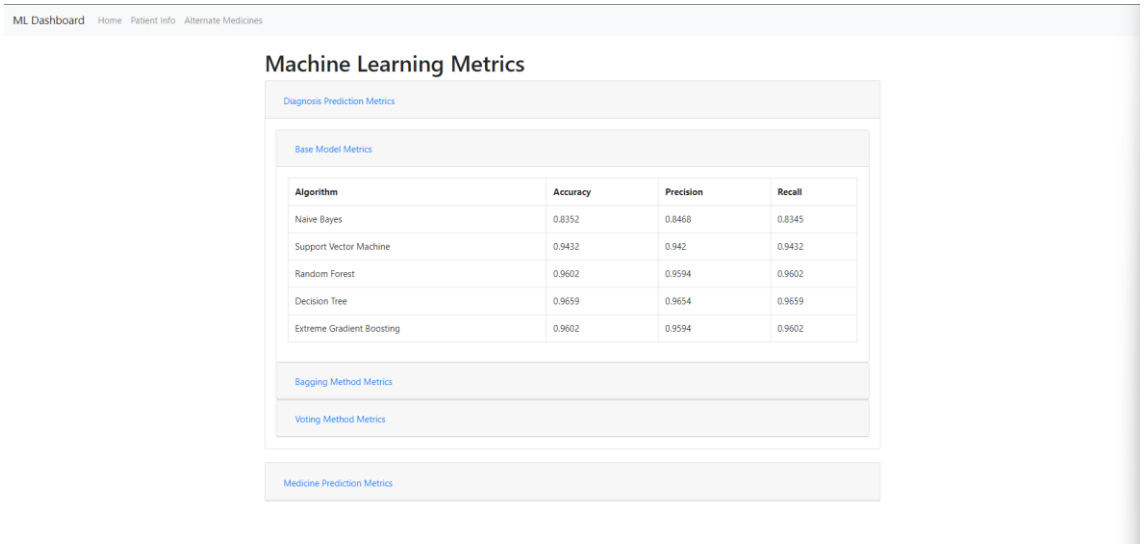
```
patient_df['diagnosa_Label'] = majority_diagnosis_label

    # List to store medicine predictions and their
probabilities
    medicine_predictions = []
    medicine_probabilities = []

   for med_model in medicine_models:
```

```
        medicine_prediction = med_model.predict(patient_df)[0]
        medicine_predictions.append(reverse_medicine_mapping[m
edicine_prediction])

        if hasattr(med_model, 'predict_proba'):
            probs = med_model.predict_proba(patient_df)[0]
            medicine_probabilities.append(probs[medicine_predi
ction] * 100)
        else:
            votes = np.array([med_model.predict(patient_df)[0]
for _ in range(100)])
            class_votes = np.bincount(votes)
            predicted_class_votes =
class_votes[medicine_prediction]
            total_votes = len(votes)
            medicine_probabilities.append((predicted_class_vot
es / total_votes) * 100)

    return render_template('results.html',
                           diagnosis_predictions=diagnosis_pre
dictions,
                           diagnosis_probabilities=diagnosis_p
robabilities,
                           medicine_predictions=medicine_predi
ctions,
                           medicine_probabilities=medicine_pro
babilities)
```

Proses prediksi obat kemudian dilanjutkan dengan pengambilan nilai probabilitas dari prediksi yang dilakukan. Program kemudian akan melakukan proses pengambilan probabilitas dari prediksi yang dilakukan dengan menerapkan proses kalkulasi yang sama dengan prediksi diagnosis.

4.2.2.2. Implementasi dari *front-end* program

4.2.2.2.1. *Home page*



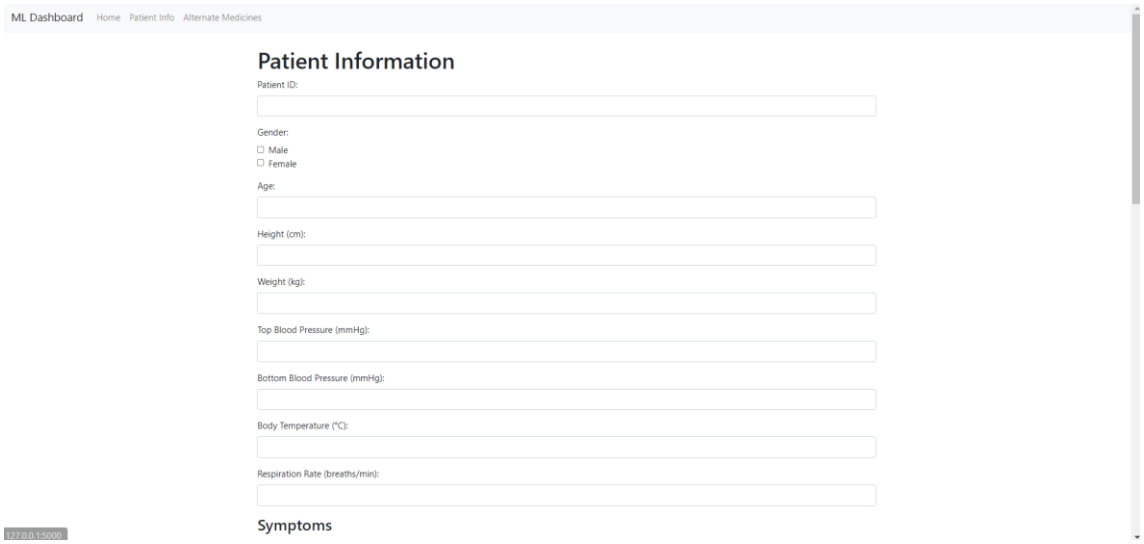

Gambar 4.1. *Home Page* dari Aplikasi Berbasis Web

*Home Page* yang dibuat dikerjakan dalam bahasa HTML yang berisikan statistika dari hasil proses *fitting* dan *testing* dari algoritma Machine Learning. *Homepage* dibagi kedalam 2 segmen, yang pertama nilai terhadap prediksi diagnosis dan yang kedua nilai terhadap prediksi obat. Masing – masing segmen dibagi kedalam 3 bagian, yakni hasil prediksi berdasarkan *base model*, Bagging Method pada semua *base model*, dan juga Voting Mehtod terhadap semua *base model* dan juga Bagging Method.

Index.html mengambil data yang disediakan pada app.py pada bagian index() yang berisikan nilai penilaian terhadap hasil dari model Machine Learning yang digunakan.

4.2.2.2.2. *Patient info page*

*Patient info page* merupakan page yang digunakan sebagai *page* pengambilan input dari data yang akan diujikan terhadap model yang telah dipilih dalam pengambilan prediksi.

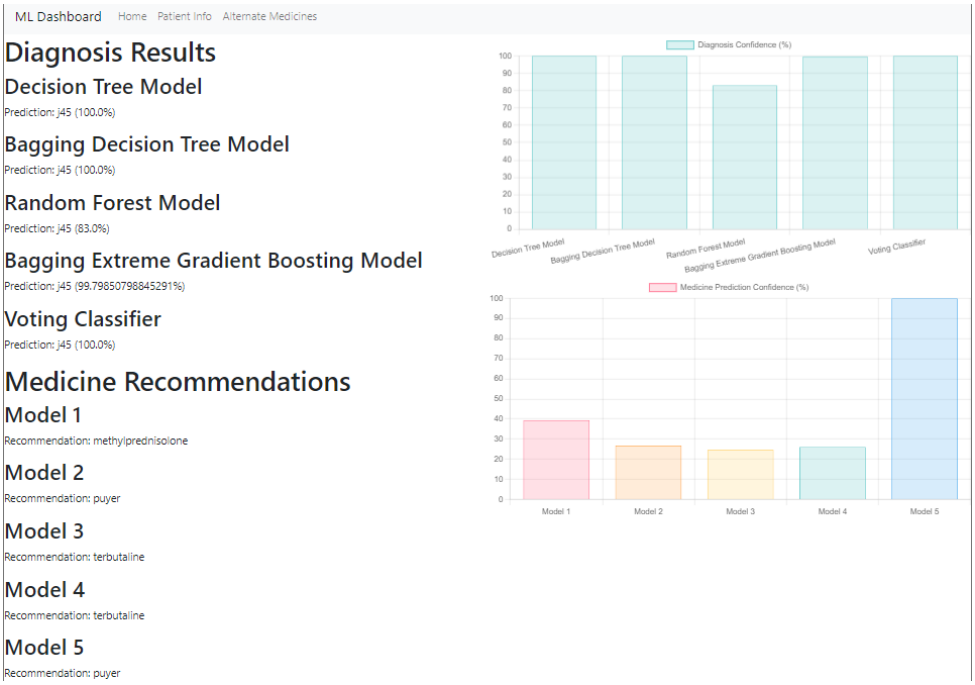



Gambar 4.2. *Patient Info Page* dari Aplikasi Berbasis Web untuk Menerima Input Karakteristik dan Gejala Pasien

*patient info page* disimpan dalam *file* HTML dengan nama 'patient_info.html'. Patient_info.html sendiri terdiri dari 3 bagian, *navbar* yang bertujuan sebagai pengatur navigasi dari program, input bagi karakteristik pasien yang terdiri dari gender, umur, tinggi dan berat badan, tekanan darah atas dan bawah, suhu badan, dan rate pernapasan, dan juga input gejala sesuai dengan yang telah dibahas pada ruang lingkup (poin 1.4.6.2). data kemudian akan diteruskan kedalam *route submit_patient_info* (segmen 4.33) untuk diolah dan diprediksi oleh model yang digunakan.

4.2.2.2.3. *Result page*

*Result page* disimpan dalam file 'result.html' yang berguna untuk menyimpan hasil prediksi dari model berserta nilai probabilitas dari prediksi yang dilakukan.



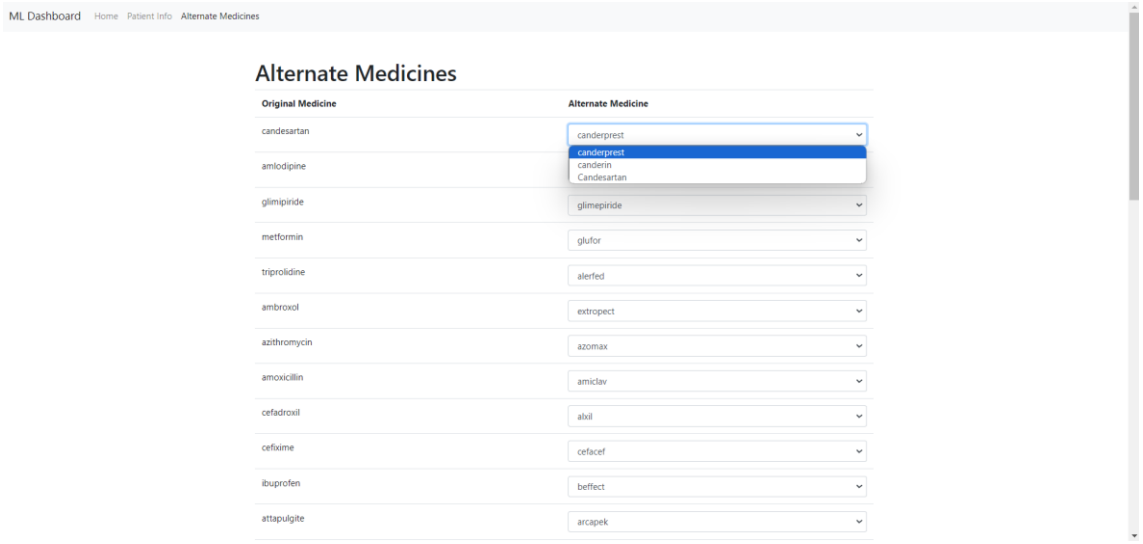

Gambar 4.3. *Result Page* untuk Menampilkan Hasil dari Prediksi Berdasarkan Data yang Di-*Input*-kan

*Result page* bekerja dengan mengambil data dari *route patient_info_submit* yang telah di-*decode* dari *label encoder* sehingga menghasilkan diagnosis dan juga obat dalam bentuk String dan bukan bilangan yang mewakili. Selain prediksi, result.html juga mengambil persentase dari probabilitas yang dihasilkan dari proses prediksi dan memvisualisasikan data probabilitas menggunakan *bar graph*.

Result.html dibagi kedalam 2 bagian dengan masing-masing bagian berisikan hasil dari prediksi diagnosis dan prediksi obat. Hasil dari prediksi ditampilkan di bagian kiri dengan probabilitas dari prediksi direpresentasikan oleh *bar graph* di bagian kanan *page*.

4.2.2.2.4. *Alternate medicine page*

*Alternate medicine page* berisikan obat alternatif yang dabat diberikan kepada pasien sesuai dengan klasifikasi obat yang dilakukan pada *data cleaning* dalam proses prediksi obat.




Gambar 4.4. *Alternative Medicine Page* untuk Menampilkan Obat Pengganti yang Dapat Diberikan Kepada Pasien.

*Alternate medicine page* mengambil data 'replacement' yang dimodifikasi dengan menghapus beberapa data yang hanya mengalami kesalahan penulisan, dan menampilkanya sebagai tabel yang berisikan obat yang diprediksi dan obat penggantinya. Obat yang diprediksi akan ditampilkan di kiri dengan obat pengganti dimasukan ke dalam *drop box* yang dapat ditampilkan dengan meng-klik *drop box* tersebut.