

## **PHASE 4: DEVELOPMENT PHASE PART 2**

<b>COURSE</b>	Artificial Intelligence
<b>PROJECT</b>	AI Based Diabetes Prediction System
<b>DATE</b>	26-10-2023

### **INTRODUCTION:**

In this phase, we transition from data preparation to the creation and fine-tuning of machine learning models. We carefully select and implement models, including an ensemble approach with a Voting Classifier, leveraging the combined strength of Random Forest, Logistic Regression, and Support Vector Machine (SVM). By splitting the dataset into training and testing sets, we ensure that our models can learn from the data while also being rigorously evaluated. The success of this phase, as indicated by model accuracy, marks a significant step toward delivering a reliable and impactful tool for predicting diabetes risk. With the models trained and ready, we move forward with confidence in our pursuit of precise and accessible diabetes risk predictions.

### **DEVELOPMENT STEPS:**

This phase is where you'll leverage our preprocessed data to create, train, and evaluate machine learning models. Here are the general steps to get started with model building and training :

- Model Training
- Model Accuracy

#### **1.MODEL TRAINING:**

Before you can train our machine learning models, you need to import them. Here's how we can import the necessary machine learning models for our AI-based Diabetes Prediction System.

Code:

```
#for numerical operations
import pickle

#for visualization
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

Here is the quick overview of the imported statements;

- **import pickle:** Used for saving and loading Python objects, including machine learning models, to and from files.
- **from sklearn.ensemble import RandomForestClassifier, VotingClassifier:** Imports machine learning models, **RandomForestClassifier** for handling complex data and **VotingClassifier** for combining multiple classifier predictions.
- **from sklearn.linear\_model import LogisticRegression:** Imports **LogisticRegression**, a simple yet effective linear classification algorithm commonly used in binary classification.
- **from sklearn.svm import SVC:** Imports **SVC** (Support Vector Classifier), a powerful classification algorithm for finding optimal decision boundaries in binary and multi-class classification tasks.

As we started importing the models, the next phase is creating the instance of models,

Code:

```
# Create instances of the models

rf = RandomForestClassifier(n_estimators=200)
lr = LogisticRegression()
svm = SVC(probability=True)
```

In the code snippet you provided, you're creating instances of machine learning models. Here's what each line does:

- **rf = RandomForestClassifier(n\_estimators=200):** This line creates an instance of the **RandomForestClassifier** model. It specifies that the random forest should consist of 200 decision trees (you can adjust this number as needed). This model is suitable for handling complex data and is robust against overfitting.
- **lr = LogisticRegression():** Here, you're creating an instance of the **LogisticRegression** model. This model is a straightforward yet effective choice for linear classification tasks, particularly in binary classification. It's known for its interpretability and computational efficiency.
- **svm = SVC(probability=True):** This line creates an instance of the **SVC** (Support Vector Classifier) model with the **probability** parameter set to **True**. The **probability**

parameter allows the model to predict probabilities, which is often useful in classification tasks. SVMs are known for their ability to find optimal decision boundaries and handle high-dimensional data.

Code:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=0)
```

After running this code, **X\_train** and **Y\_train** will contain the features and target variable for the training set, while **X\_test** and **Y\_test** will contain the features and target variable for the testing set. This allows us to train our machine learning models on the training set and evaluate their performance on the testing set.

As our model is splitted and trained ,the next thing we can do is creating a voting classifier and fit our model into them .

Code:

```
# Create a voting classifier
voting_classifier = VotingClassifier(estimators=[('rf', rf), ('lr', lr), ('svm', svm)],
voting='soft')

# Fit the voting classifier on the training data
print(voting_classifier.fit(X_train, Y_train))
```

Output:

```
✓ [17] # Create a voting classifier
0s voting_classifier = VotingClassifier(estimators=[('rf', rf), ('lr', lr), ('svm', svm)], voting='soft')

✓ 1s # Fit the voting classifier on the training data
print(voting_classifier.fit(X_train, Y_train))

VotingClassifier(estimators=[('rf', RandomForestClassifier(n_estimators=200)),
('lr', LogisticRegression()),
('svm', SVC(probability=True))],
voting='soft')
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
```

## 2.MODEL ACCURACY:

Moving on to model accuracy is an important step to evaluate how well your machine learning models are performing. We can assess the model accuracy using various metrics. The accuracy score is a common metric to measure the overall correctness of your model's predictions.

### Code:

```
# Evaluate the accuracy of the voting classifier on the test data

accuracy = voting_classifier.score(X_test, Y_test)
print("Accuracy of Voting Classifier:", accuracy)
```

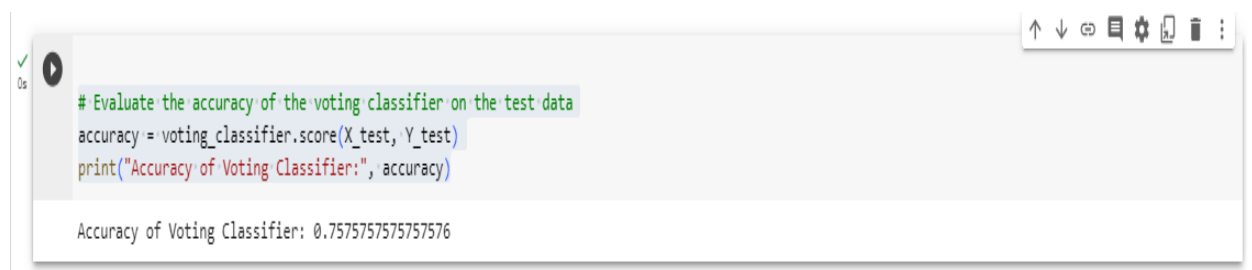
Here's a brief explanation:

- **voting\_classifier.score(X\_test, Y\_test):** The **score** method of the **voting\_classifier** object is used to evaluate the model's accuracy on the provided test data. **X\_test** contains the features, and **Y\_test** contains the true target labels.
- **accuracy = ...:** The result of the accuracy calculation is assigned to the variable **accuracy**.
- **print("Accuracy of Voting Classifier:", accuracy):** This line prints the accuracy of the Voting Classifier on the test data to the console.

The accuracy score represents the proportion of correct predictions made by our model on the test data. It's a common metric to assess the model's performance in classification tasks. The accuracy value will be between 0 (no correct predictions) and 1 (all predictions are correct).

In our case, the accuracy value will tell you how well the Voting Classifier is performing in predicting diabetes based on the test data.

### Output:



The screenshot shows a Jupyter Notebook interface. At the top, there is a toolbar with icons for undo, redo, run, and other actions. Below the toolbar, the code from the previous block is pasted into a code cell. The code is: 

```
# Evaluate the accuracy of the voting classifier on the test data
accuracy = voting_classifier.score(X_test, Y_test)
print("Accuracy of Voting Classifier:", accuracy)
```

 The code is highlighted in a light blue color. Below the code, the output of the code is displayed: 

```
Accuracy of Voting Classifier: 0.7575757575757576
```

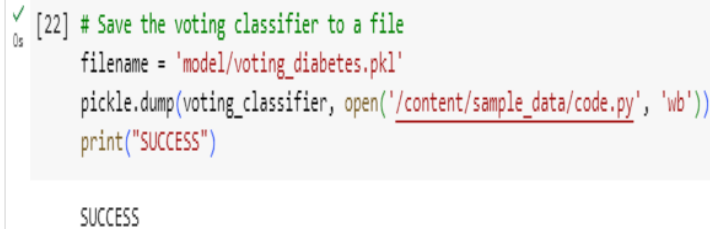
### Code:

```
# Save the voting classifier to a file
filename = 'model/voting_diabetes.pkl'
pickle.dump(voting_classifier, open('/content/sample_data/code.py', 'wb'))
print("SUCCESS")
```

Here's a summary of what the code does:

- It specifies the file path where the model will be saved using the **filename** variable.
- It saves the **voting\_classifier** model to the specified file path using **pickle.dump()**.
- It prints a success message indicating that the model has been saved.

### Output:



```
✓ [22] # Save the voting classifier to a file
      filename = 'model/voting_diabetes.pkl'
      pickle.dump(voting_classifier, open('/content/sample_data/code.py', 'wb'))
      print("SUCCESS")

SUCCESS
```

We can now proceed to the next steps in your project, such as model deployment or further analysis, knowing that the model has been successfully saved.

### CHECK OUT THE CODE:

[https://colab.research.google.com/drive/1llxazFt1201IWkvuMcy29G0KOArcgw6u#scrollTo=13VKf1\\_reJSu](https://colab.research.google.com/drive/1llxazFt1201IWkvuMcy29G0KOArcgw6u#scrollTo=13VKf1_reJSu)

### CONCLUSION:

In this phase, we've selected and implemented machine learning models, including a Voting Classifier that combines the strengths of different classifiers, divided our dataset into training and testing sets, trained our models, evaluated their performance, and successfully saved the trained Voting Classifier to a file. With a focus on accuracy and robustness, our project is now one step closer to delivering a valuable tool for healthcare professionals and individuals seeking accurate diabetes risk predictions. This model represents a crucial component in our journey toward deploying a practical and impactful system.