

```
from IPython.display import Image, display
display(Image(filename='Lecture 7.png'))
```



Import necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

## pip install imblearn
```

Imbalanced dataset

An imbalanced dataset occurs when the number of samples in each class is uneven, causing machine learning models to be biased toward the majority class.

Cerebral Stroke Prediction-Imbalanced Dataset — A dataset focused on stroke prediction, explicitly described as imbalanced. Kaggle

Link: Kaggle – Cerebral Stroke Prediction-Imbalanced Dataset

<https://www.kaggle.com/datasets/shashwatwork/cerebral-stroke-predictionimbalaced-dataset/data>

```
data = pd.read_csv('dataset.csv')
```

```
data.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	\
0	30669	Male	3.0	0	0	No	
1	30468	Male	58.0	1	0	Yes	
2	16523	Female	8.0	0	0	No	
3	56543	Female	70.0	0	0	Yes	
4	46136	Male	14.0	0	0	No	

	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	\
0	children	Rural	95.12	18.0	NaN	
1	Private	Urban	87.96	39.2	never	
2	Private	Urban	110.89	17.6	NaN	
3	Private	Rural	69.04	35.9	formerly	
4	Never_worked	Rural	161.28	19.1	NaN	

	stroke
0	0
1	0
2	0
3	0
4	0

```
le = LabelEncoder()
```

```
data['gender'] = le.fit_transform(data['gender'])
```

```
data['ever_married'] = le.fit_transform(data['ever_married'])
```

```
data['work_type'] = le.fit_transform(data['work_type'])
```

```
data['Residence_type'] = le.fit_transform(data['Residence_type'])
```

```
data['stroke'].value_counts()
```

stroke	
0	42617

```

1      783
Name: count, dtype: int64

data.isnull().sum()

id                0
gender            0
age              0
hypertension      0
heart_disease     0
ever_married      0
work_type         0
Residence_type    0
avg_glucose_level 0
bmi              1462
smoking_status    13292
stroke            0
dtype: int64

len(data)

43400

X = data.drop(['id', 'stroke', 'bmi', 'smoking_status'], axis=1).values
# Replace 'classification' with your target column
y = data['stroke'].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

#####
##

```

1. Random Oversampling

Random Oversampling is a data-level technique used to handle class imbalance by increasing the number of samples in the minority class.

□ How It Works

- The algorithm looks at the minority class.
- Randomly selects samples from this class (with replacement).
- Duplicates them until both classes have similar numbers of samples.

```

from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler()

```

```

X_resampled, y_resampled = ros.fit_resample(X_train, y_train)

print("Before:")
pd.Series(y_train).value_counts()

Before:
0    34088
1     632
Name: count, dtype: int64

print("After:")
pd.Series(y_resampled).value_counts()

After:
0    34088
1    34088
Name: count, dtype: int64

#####

```

2. Random Undersampling

Random Undersampling is a technique used to handle imbalanced datasets by reducing the number of samples in the majority class.

How It Works

Identify the majority class.

Randomly remove samples from this class.

Keep removing until the class distribution becomes balanced.

```

from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler()

X_rus, y_rus = rus.fit_resample(X_train, y_train)

print("After Undersampling:", pd.Series(y_rus).value_counts())

After Undersampling: 0    632
1    632
Name: count, dtype: int64

#####

```

3. Synthetic Minority Over-sampling Technique (SMOTE) Oversampling

SMOTE is an oversampling method that creates new synthetic samples for the minority class instead of duplicating them.

It generates new samples by interpolating between existing minority class samples.

How SMOTE Works (Intuition)

For each minority class sample:

- Find its k nearest neighbors (usually k=5).
- Pick one neighbor randomly.
- Create a new synthetic sample on the line between the two points.

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(sampling_strategy=0.5) # minority = 50% of majority
X_sm, y_sm = sm.fit_resample(X_train, y_train)

print("After SMOTE:", pd.Series(y_sm).value_counts())

After SMOTE: 0    34088
            1    17044
Name: count, dtype: int64

#####
##
```

4. SMOTE + Tomek Links (Hybrid)

This method combines oversampling (SMOTE) with undersampling (Tomek Links) to improve data quality and prevent noise.

```
from imblearn.combine import SMOTETomek

smt = SMOTETomek()
X_smt, y_smt = smt.fit_resample(X_train, y_train)

print("\nAfter SMOTE + Tomek Links:")
print(pd.Series(y_smt).value_counts())

After SMOTE + Tomek Links:
0    33932
```

```
1      33932
Name: count, dtype: int64
```

Evaluation models

```
def evaluate_model(model_name, y_true, y_pred):

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    # Create a report
    report = classification_report(y_true, y_pred)

    # Output results
    metrics = {
        'Model Name': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,

        'Classification Report': report
    }

    # Plot Confusion Matrix
    plt.figure(figsize=(4, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=np.unique(y_true),
                yticklabels=np.unique(y_true))
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

    return metrics
```

Random Forest Classifier

```
# Function to train + evaluate Random Forest
def evaluate_model(Xt, yt, method_name):
    print(f"\n===== {method_name} =====")
```

```

rf = RandomForestClassifier(random_state=42)
rf.fit(Xt, yt)

preds = rf.predict(X_test)
probs = rf.predict_proba(X_test)[:, 1]

print(classification_report(y_test, preds))
print("ROC-AUC:", roc_auc_score(y_test, probs))

# -----
# 1. Original (Imbalanced)
# -----
evaluate_model(X_train, y_train, "Original Imbalanced Data")

# -----
# 2. Random Oversampling
# -----
ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_resample(X_train, y_train)
evaluate_model(X_ros, y_ros, "Random Oversampling")

# -----
# 3. Random Undersampling
# -----
rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(X_train, y_train)
evaluate_model(X_rus, y_rus, "Random Undersampling")

# -----
# 4. SMOTE
# -----
smote = SMOTE(random_state=42)
X_sm, y_sm = smote.fit_resample(X_train, y_train)
evaluate_model(X_sm, y_sm, "SMOTE Oversampling")

smt = SMOTETomek()
X_smt, y_smt = smt.fit_resample(X_train, y_train)
evaluate_model(X_smt, y_smt, "Tomek Links")

```

```

===== Original Imbalanced Data =====
      precision    recall  f1-score   support

0               0.98         1.00         0.99         8523

```

1	0.11	0.02	0.03	157
accuracy			0.98	8680
macro avg	0.55	0.51	0.51	8680
weighted avg	0.97	0.98	0.97	8680

ROC-AUC: 0.7136227861515226

===== Random Oversampling =====

	precision	recall	f1-score	support
0	0.98	0.99	0.99	8523
1	0.06	0.03	0.04	157
accuracy			0.97	8680
macro avg	0.52	0.51	0.51	8680
weighted avg	0.97	0.97	0.97	8680

ROC-AUC: 0.7324067285897806

===== Random Undersampling =====

	precision	recall	f1-score	support
0	0.99	0.73	0.84	8523
1	0.05	0.79	0.10	157
accuracy			0.73	8680
macro avg	0.52	0.76	0.47	8680
weighted avg	0.98	0.73	0.83	8680

ROC-AUC: 0.8271989394003936

===== SMOTE Oversampling =====

	precision	recall	f1-score	support
0	0.98	0.99	0.99	8523
1	0.05	0.02	0.03	157
accuracy			0.98	8680
macro avg	0.52	0.51	0.51	8680
weighted avg	0.97	0.98	0.97	8680

ROC-AUC: 0.7035985803868288

===== Tomek Links =====

	precision	recall	f1-score	support
0	0.98	0.99	0.99	8523
1	0.06	0.02	0.03	157
accuracy			0.98	8680

macro avg	0.52	0.51	0.51	8680
weighted avg	0.97	0.98	0.97	8680

ROC-AUC: 0.698356862771474

```
# -----
def evaluate_model(Xt, yt, method_name):
    print(f"\n===== {method_name} =====")

    svm = SVC(kernel='rbf', probability=True, random_state=42)
    svm.fit(Xt, yt)

    preds = svm.predict(X_test)
    probs = svm.predict_proba(X_test)[:, 1]

    print(classification_report(y_test, preds))
    print("ROC-AUC:", roc_auc_score(y_test, probs))

# -----
# 1. Original (Imbalanced)
# -----
evaluate_model(X_train, y_train, "Original Imbalanced Data")

# -----
# 2. Random Oversampling
# -----
ros = RandomOverSampler(random_state=42)
X_ros, y_ros = ros.fit_resample(X_train, y_train)
evaluate_model(X_ros, y_ros, "Random Oversampling")

# -----
# 3. Random Undersampling
# -----
rus = RandomUnderSampler(random_state=42)
X_rus, y_rus = rus.fit_resample(X_train, y_train)
evaluate_model(X_rus, y_rus, "Random Undersampling")

# -----
# 4. SMOTE Oversampling
# -----
smote = SMOTE(random_state=42)
X_sm, y_sm = smote.fit_resample(X_train, y_train)
evaluate_model(X_sm, y_sm, "SMOTE Oversampling")

# -----
```

5. SMOTE + Tomek Links

```
smt = SMOTETomek(random_state=42)
X_smt, y_smt = smt.fit_resample(X_train, y_train)
evaluate_model(X_smt, y_smt, "SMOTE + Tomek Links")
```

===== Original Imbalanced Data =====

C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])

C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])

C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])

	precision	recall	f1-score	support
0	0.98	1.00	0.99	8523
1	0.00	0.00	0.00	157
accuracy			0.98	8680
macro avg	0.49	0.50	0.50	8680
weighted avg	0.96	0.98	0.97	8680

ROC-AUC: 0.7358761717077282

===== Random Oversampling =====

	precision	recall	f1-score	support
0	1.00	0.73	0.84	8523
1	0.05	0.84	0.10	157
accuracy			0.73	8680
macro avg	0.52	0.78	0.47	8680
weighted avg	0.98	0.73	0.83	8680

ROC-AUC: 0.8595348218496075

```
===== Random Undersampling =====
      precision    recall  f1-score   support
```

```
    0          1.00        0.72        0.83        8523
    1          0.05        0.83        0.10         157
```

```
 accuracy          0.72        8680
macro avg          0.52        0.78        0.47        8680
weighted avg       0.98        0.72        0.82        8680
```

ROC-AUC: 0.8547467287840844

```
===== SMOTE Oversampling =====
      precision    recall  f1-score   support
```

```
    0          1.00        0.72        0.84        8523
    1          0.05        0.85        0.10         157
```

```
 accuracy          0.72        8680
macro avg          0.52        0.79        0.47        8680
weighted avg       0.98        0.72        0.82        8680
```

ROC-AUC: 0.8608915852272345

```
===== SMOTE + Tomek Links =====
      precision    recall  f1-score   support
```

```
    0          1.00        0.72        0.84        8523
    1          0.05        0.85        0.10         157
```

```
 accuracy          0.72        8680
macro avg          0.52        0.79        0.47        8680
weighted avg       0.98        0.72        0.82        8680
```

ROC-AUC: 0.8609409084896544

```
#####
####
```

If you download the dataset, it includes default train and test files.

```
train = pd.read_csv('train.csv')
test=pd.read_csv('test.csv')

X_train = train.drop(['target'], axis=1)
y_train = train['target'].values

X_test = test.drop(['target'], axis=1)
y_test = test['target'].values
```

Ensemble learning

Ensemble learning is a machine learning technique where multiple models (often called “weak learners”) are combined to create a stronger, more accurate model.

```
def evaluate_model(model_name, y_true, y_pred):  
  
    # Calculate metrics  
    accuracy = accuracy_score(y_true, y_pred)  
    precision = precision_score(y_true, y_pred, average='weighted')  
    recall = recall_score(y_true, y_pred, average='weighted')  
    f1 = f1_score(y_true, y_pred, average='weighted')  
    cm = confusion_matrix(y_true, y_pred)  
  
    # Create a report  
    report = classification_report(y_true, y_pred)  
  
    # Output results  
    metrics = {  
        'Model Name': model_name,  
        'Accuracy': accuracy,  
        'Precision': precision,  
        'Recall': recall,  
        'F1 Score': f1,  
  
        'Classification Report': report  
    }  
  
    # Plot Confusion Matrix  
    plt.figure(figsize=(4, 4))  
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,  
                xticklabels=np.unique(y_true),  
                yticklabels=np.unique(y_true))  
    plt.title(f'Confusion Matrix for {model_name}')  
    plt.xlabel('Predicted Label')  
    plt.ylabel('True Label')  
    plt.show()  
  
    return metrics
```

Types of Ensemble Learning

1. Bagging (Bootstrap Aggregating)

Trains several models independently on random subsets of the data.

Their predictions are averaged (regression) or voted (classification).

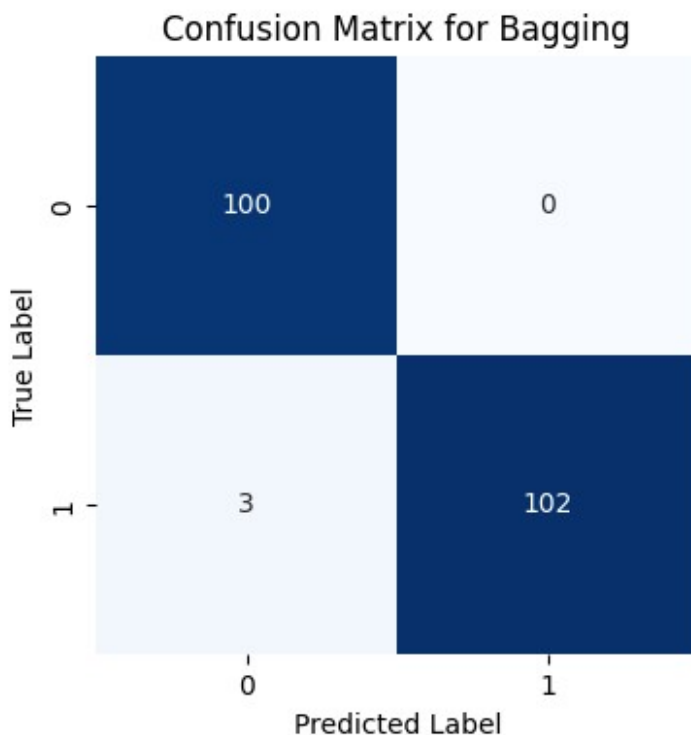
```

from sklearn.ensemble import BaggingClassifier
# Bagging ensemble
bagging = BaggingClassifier(
    estimator=DecisionTreeClassifier(), # updated parameter
    n_estimators=20,
    max_samples=0.8,
    bootstrap=True
)

bagging.fit(X_train, y_train)
y_pred = bagging.predict(X_test)

evaluation_results = evaluate_model('Bagging', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
              f"{key}: \n{value}")

```



Model Name:
 Bagging
 Accuracy: 0.9854
 Precision: 0.9858
 Recall: 0.9854

F1 Score: 0.9854				
	precision	recall	f1-score	support
0	0.97	1.00	0.99	100
1	1.00	0.97	0.99	105
accuracy			0.99	205
macro avg	0.99	0.99	0.99	205
weighted avg	0.99	0.99	0.99	205

2. Boosting

Models are trained sequentially, where each model corrects the errors of the previous one.

Produces a strong model from many weak ones.

Examples: AdaBoost, Gradient Boosting, XGBoost, LightGBM, CatBoost.

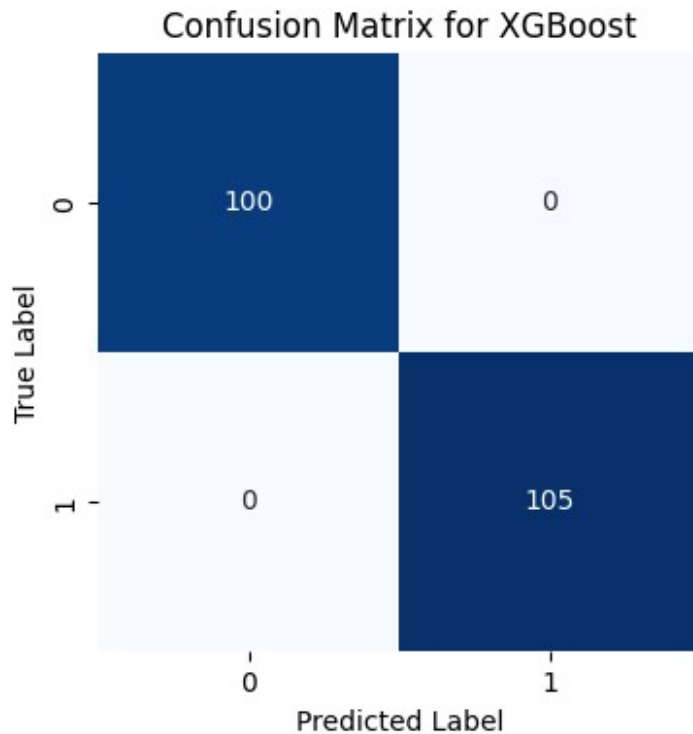
```
### pip install xgboost
import xgboost as xgb

model = xgb.XGBClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=4,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='binary:logistic',
    eval_metric='logloss'
)

# Train
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

evaluation_results = evaluate_model('XGBoost', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
              f"{key}: \n{value}")
```



Model Name:

XGBoost

Accuracy: 1.0000

Precision: 1.0000

Recall: 1.0000

F1 Score: 1.0000

	precision	recall	f1-score	support
0	1.00	1.00	1.00	100
1	1.00	1.00	1.00	105
accuracy			1.00	205
macro avg	1.00	1.00	1.00	205
weighted avg	1.00	1.00	1.00	205

AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=100, learning_rate=0.5)
ada.fit(X_train, y_train)
y_pred = ada.predict(X_test)
```

```
evaluation_results = evaluate_model('AdaBoost', y_test, y_pred)
```

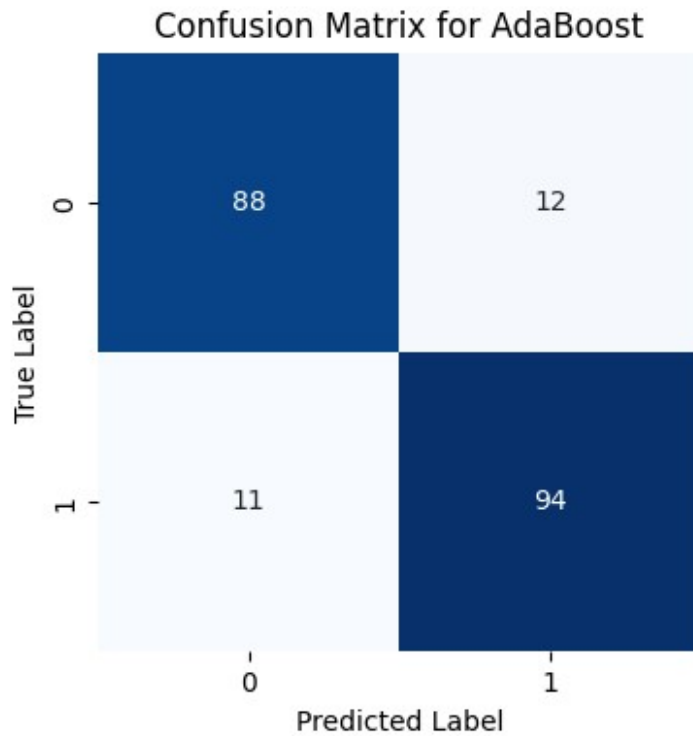
Print the evaluation results

```
for key, value in evaluation_results.items():
```

```

if key == 'Classification Report':
    print(value) # Print report separately for better readability
else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```



Model Name:

AdaBoost

Accuracy: 0.8878

Precision: 0.8878

Recall: 0.8878

F1 Score: 0.8878

	precision	recall	f1-score	support
0	0.89	0.88	0.88	100
1	0.89	0.90	0.89	105
accuracy			0.89	205
macro avg	0.89	0.89	0.89	205
weighted avg	0.89	0.89	0.89	205

```

from sklearn.ensemble import RandomForestClassifier, VotingClassifier
# Base models
model1 = LogisticRegression(max_iter=200)
model2 = SVC(probability=True)

```

```

model3 = RandomForestClassifier()

# Voting ensemble
ensemble = VotingClassifier(
    estimators=[("lr", model1), ("svm", model2), ("rf", model3)],
    voting="soft"
)

# Train
ensemble.fit(X_train, y_train)

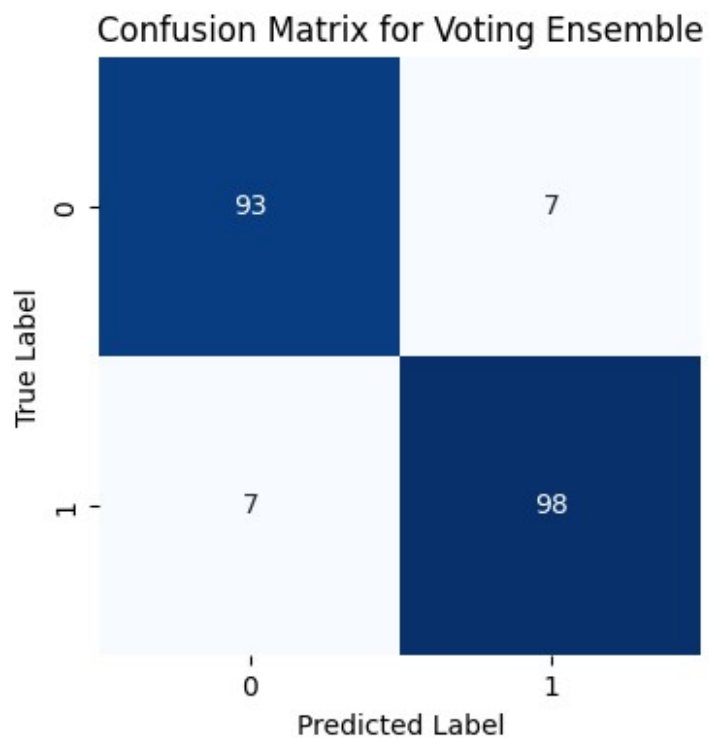
# Predict
y_pred = ensemble.predict(X_test)

evaluation_results = evaluate_model('Voting Ensemble', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
              f"{key}: \n{value}")

C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\
linear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to
converge after 200 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence
(max_iter=200).
You might also want to 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(

```



Model Name:

Voting Ensemble

Accuracy: 0.9317

Precision: 0.9317

Recall: 0.9317

F1 Score: 0.9317

	precision	recall	f1-score	support
0	0.93	0.93	0.93	100
1	0.93	0.93	0.93	105
accuracy			0.93	205
macro avg	0.93	0.93	0.93	205
weighted avg	0.93	0.93	0.93	205