

Міністерство освіти і науки України
Національний технічний університет України “Київський політехнічний
інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №3

з дисципліни

“Програмування інтелектуальних інформаційних систем”

Виконала

ІІ-21 Скрипець О. О.

Перевірив

Баришич Л. М.

Київ 2024

Завдання

1. Пройти тутор:

<https://www.kaggle.com/code/jhoward/linear-model-and-neural-net-from-scratch#Deep-learning>

2. Побудувати рендом форест звідси:

<https://www.kaggle.com/code/jhoward/how-random-forests-really-work/>

2.1. Натрейнити на датасеті звідси: `'/kaggle/input/car-evaluation-data-set/car_evaluation.csv'`

Class - залежна змінна

Важливо! Не забудьте енкодер

```
encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors',  
'persons', 'lug_boot', 'safety'])
```

2.2 Вивести **confusion matrix**, **auc**, **Classification report**

3 Зробити буст попередньої моделі XGBoost. Порівняти результати

<https://machinelearningmastery.com/random-forest-ensembles-with-xgboost/>




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive')
data = "/content/drive/My Drive/data/car_evaluation.csv"
df = pd.read_csv(data, header=None, sep=',\s*', engine='python')
```

 Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount

ДИВЛЮСЬ ЯК ВИГЛЯДАЄ ДАТАСЕТ

```
df.head()
```

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc


Подальші дії:

 [Переглянути рекомендовані графіки](#)

[New interactive sheet](#)

ДИВЛЮСЬ РОЗМІРИ

```
df.shape
```

 (1728, 7)

ДАЮ НАЗВИ СТОВПЦЯМ

```
df.columns=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'Class']
df.head()
```



	buying	maint	doors	persons	lug_boot	safety	Class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc



Подальші дії:

[Переглянути рекомендовані графіки](#)[New interactive sheet](#)

df.info()



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  -
0   buying      1728 non-null   object
1   maint       1728 non-null   object
2   doors       1728 non-null   object
3   persons     1728 non-null   object
4   lug_boot    1728 non-null   object
5   safety      1728 non-null   object
6   Class       1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

Переглядаю розподіл серед класів автомобілів

df['Class'].value_counts()



```
count
Class
unacc    1210
acc       384
good       69
vgood     65
```

```
X = df.drop(['Class'], axis=1)
y = df['Class']
```

Розділяємо дані на навчальні та тестові набори

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 42)
```

```
X_train.shape, X_test.shape
```

```
((1157, 6), (571, 6))
```

ВСТАНОВЛЮЮ бібліотеку

```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.4-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (1
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (1
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy:
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (1
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages
Downloading category_encoders-2.6.4-py2.py3-none-any.whl (82 kB)
82.0/82.0 kB 2.0 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.4
```


імпортую бібліотеку


```
import category_encoders as ce
```

Ініціалізую OrdinalEncoder для перетворення категоріальних змінних в числові значення

```
encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety'])
X_train = encoder.fit_transform(X_train)
X_test = encoder.transform(X_test)
```

```
X_train.head()
```



	buying	maint	doors	persons	lug_boot	safety	
48	1	1	1	1	1	1	
468	2	1	1	2	2	1	
155	1	2	1	1	2	2	
1721	3	3	2	1	2	2	
1208	4	3	3	1	2	2	

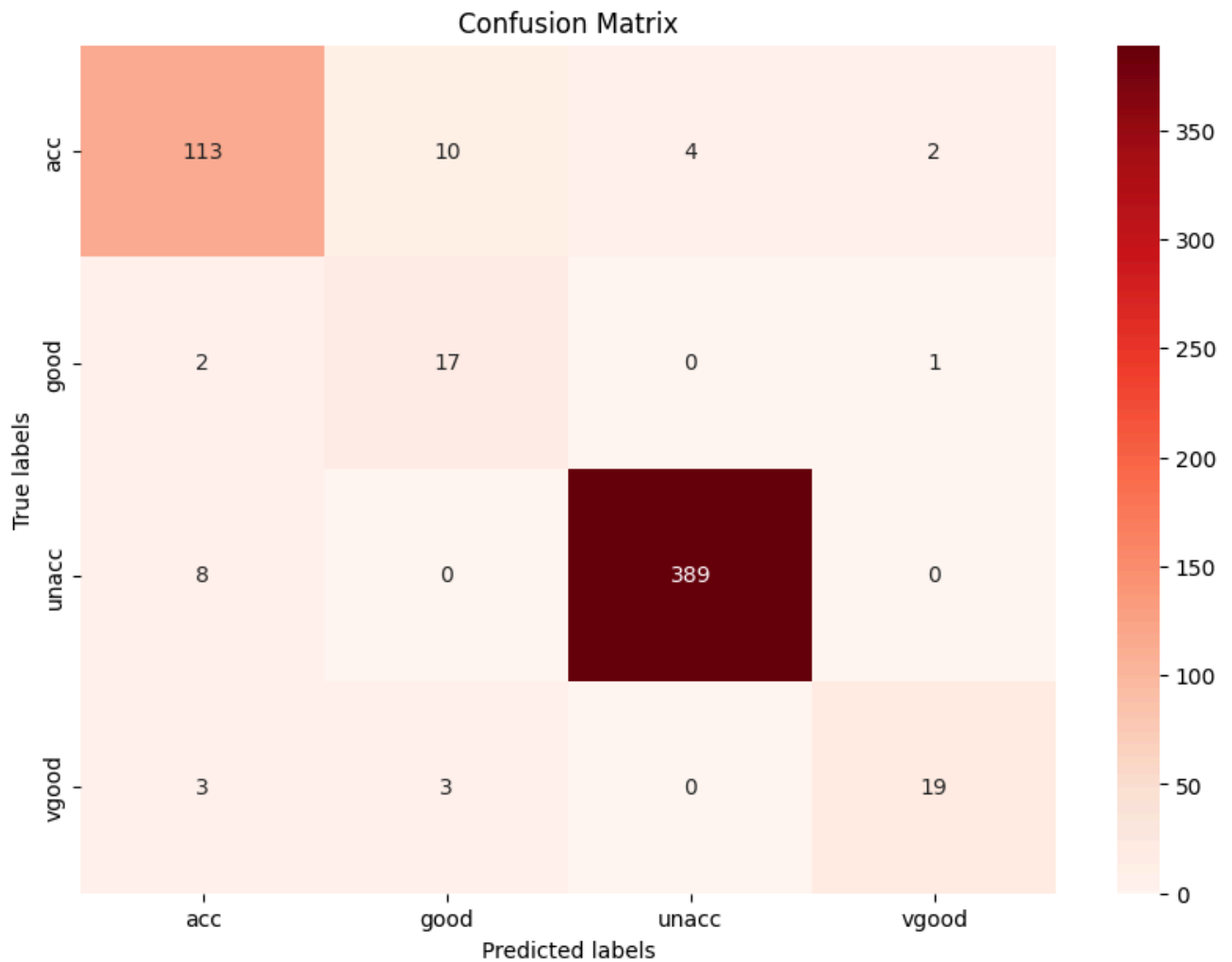
Подальші дії: [Переглянути рекомендовані графіки](#) [New interactive sheet](#)

✓ Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

Створюю екземпляр класифікатора та прогножую результати тестового набору

```
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
cm = confusion_matrix(y_test, y_pred, labels=rf_model.classes_)
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=rf_model.classes_, yticklabels=r
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



```
cr = classification_report(y_test, y_pred, target_names=rf_model.classes_)
print(cr)
```



	precision	recall	f1-score	support
acc	0.90	0.88	0.89	129
good	0.57	0.85	0.68	20
unacc	0.99	0.98	0.98	397
vgood	0.86	0.76	0.81	25
accuracy			0.94	571
macro avg	0.83	0.87	0.84	571
weighted avg	0.95	0.94	0.94	571

Точність (Precision):

Найвищу точність показує клас 'unacc' (0.99), що свідчить про рідкісні помилки моделі при прогнозуванні цього класу. Найнижча точність зафіксована у класу 'good' (0.57), що може вказувати на труднощі моделі у правильній класифікації цього класу.

Повнота (Recall):

Найвища повнота також у класу 'unacc' (0.98), що говорить про те, що модель ефективно

ідентифікує більшість випадків цього класу. Найнижча повнота у класу 'vgood' (0.76) свідчить про те, що модель пропускає деякі приклади з цього класу.

F1-оцінка (F1-score):

Найвищу F1-оцінку отримує клас 'unacc' (0.98), що вказує на хорошу збалансованість між точністю та повнотою для цього класу. Найнижча F1-оцінка у класу 'good' (0.68) свідчить про менш сприятливу збалансованість між цими двома метриками.

Загальна точність (Accuracy):

Загальна точність моделі становить 0.94, що свідчить про те, що вона правильно класифікує 94% усіх прикладів.

Середні оцінки:

Для середніх оцінок: Macro avg показує 0.83 для точності, 0.87 для повноти і 0.84 для F1-оцінки, що є середнім значенням для всіх класів. Weighted avg показує 0.95 для точності, 0.94 для повноти і 0.94 для F1-оцінки, що враховує дисбаланс класів у тестовій вибірці.

✓ Висновок:

Модель демонструє загалом високу точність і добру здатність до класифікації, особливо для класу 'unacc'. Однак існують проблеми з класифікацією класу 'good', що вимагає вдосконалення.

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

y_prob = rf_model.predict_proba(X_test)

# Ініціалізуємо словники для FPR, TPR та AUC
fpr = {}
tpr = {}
roc_auc = {}

# Обчислюємо FPR, TPR та AUC для кожного класу
for i, class_label in enumerate(rf_model.classes_):
    fpr[i], tpr[i], _ = roc_curve(y_test, y_prob[:, i], pos_label=class_label)
    roc_auc[i] = auc(fpr[i], tpr[i])

# Налаштовуємо кольори для кожного класу
colors = ['#FF5733', '#33FF57', '#3357FF', '#FF33A8'] # Задаємо нові кольори

plt.figure(figsize=(12, 8))

# Побудова графіків ROC для кожного класу
for i, class_label in enumerate(rf_model.classes_):
    plt.plot(fpr[i], tpr[i], color=colors[i], label=f'Class {class_label} (AUC = {roc_auc[i]:.2}

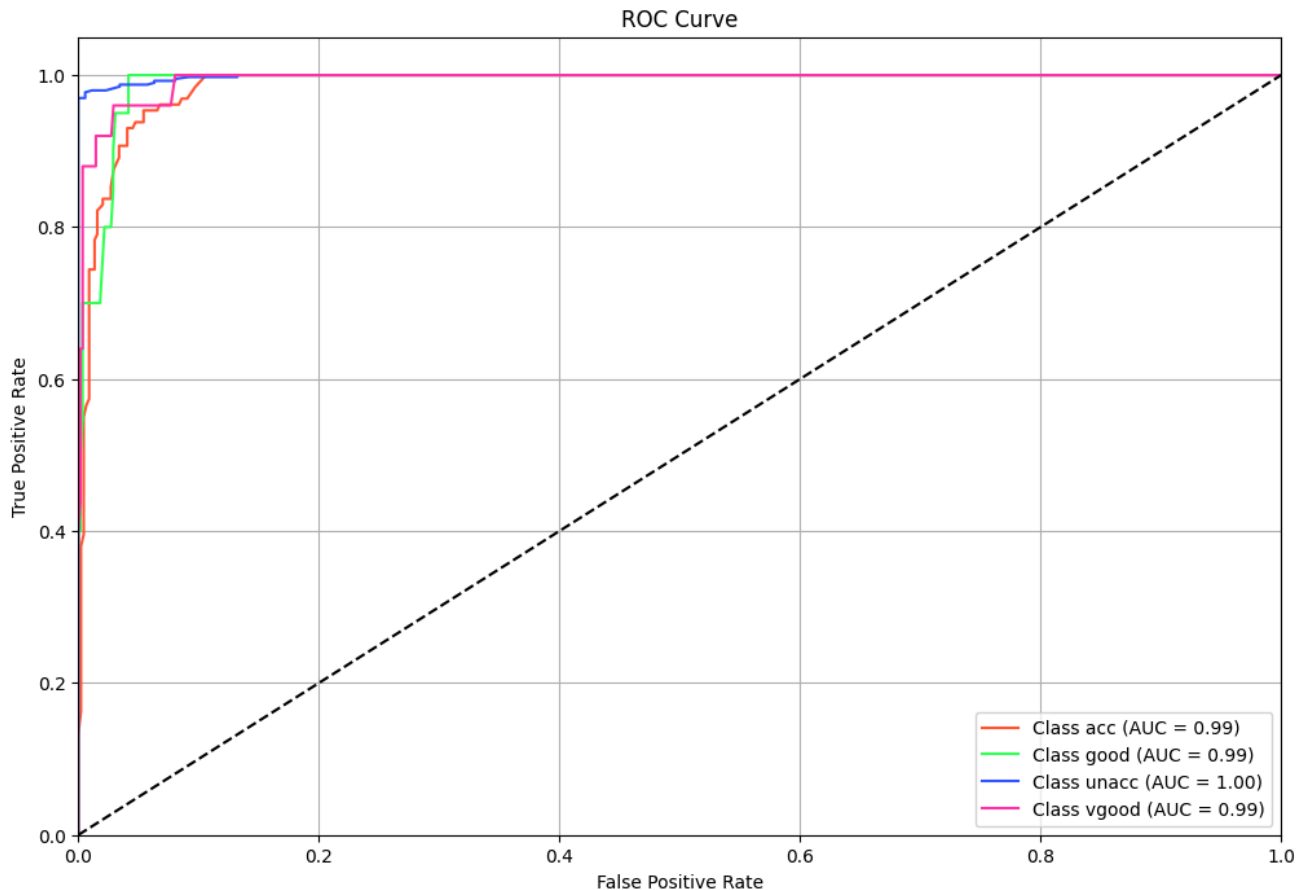
# Додаємо лінію випадкових прогнозів
plt.plot([0, 1], [0, 1], 'k--')

# Налаштовуємо осі та заголовок
plt.xlim([0.0, 1.0])
```



```
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')

# Додаємо легенду
plt.legend(loc='lower right')
plt.grid() # Додаємо сітку для кращої читабельності
plt.show()
```



```
y_prob = rf_model.predict_proba(X_test)
auc = roc_auc_score(y_test, y_prob, multi_class='ovr')
print(f'AUC: {auc}')
```



AUC: 0.9925821283497147

Ми отримали дуже високе значення AUC, що свідчить про відмінну здатність моделі розрізняти між класами. AUC вимірює площу під кривою ROC (Receiver Operating Characteristic) і показує, наскільки ефективно модель може відокремлювати різні класи.

Оптимізація гіперпараметрів моделі може значно поліпшити результати. Рекомендується експериментувати з різними значеннями для параметрів, таких як `n_estimators` (кількість дерев), `max_depth` (максимальна глибина дерев), `min_samples_split` (мінімальна кількість зразків для розділення вузла), `min_samples_leaf` (мінімальна кількість зразків у листі) тощо.

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Створюємо модель випадкового лісу
rf_model = RandomForestClassifier(random_state=42)

# Визначаємо сітку параметрів для пошуку
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Створюємо об'єкт GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, scoring='accuracy')

# Проводимо пошук найкращих параметрів
grid_search.fit(X_train, y_train)

# Виводимо найкращі параметри
print(f'Best parameters: {grid_search.best_params_}')

# Можна також вивести найкращу точність
best_accuracy = grid_search.best_score_
print(f'Best accuracy: {best_accuracy:.4f}')

```



```

Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best accuracy: 0.9464

```



Максимальна глибина дерев (`max_depth: None`) не обмежена, що дозволяє деревам рости до максимальної глибини. Це може бути корисно, але також підвищує ризик переобучення.

Мінімальна кількість зразків у листі (`min_samples_leaf: 1`) дорівнює 1, що означає, що в кожному листі дерева може бути лише один зразок.

Мінімальна кількість зразків для розділення вузла (`min_samples_split: 2`) становить 2, що дозволяє розділяти вузли, навіть якщо є лише два зразки.

Кількість дерев у лісі (`n_estimators: 200`) дорівнює 200, що забезпечує достатню кількість моделей для досягнення стабільних результатів.

Тепер ми можемо створити нову модель, використовуючи ці параметри, і навчити її на наших даних.

```
best_rf_model = grid_search.best_estimator_
```

Можемо перевірити, чи модель з новими параметрами покращила результати

```

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

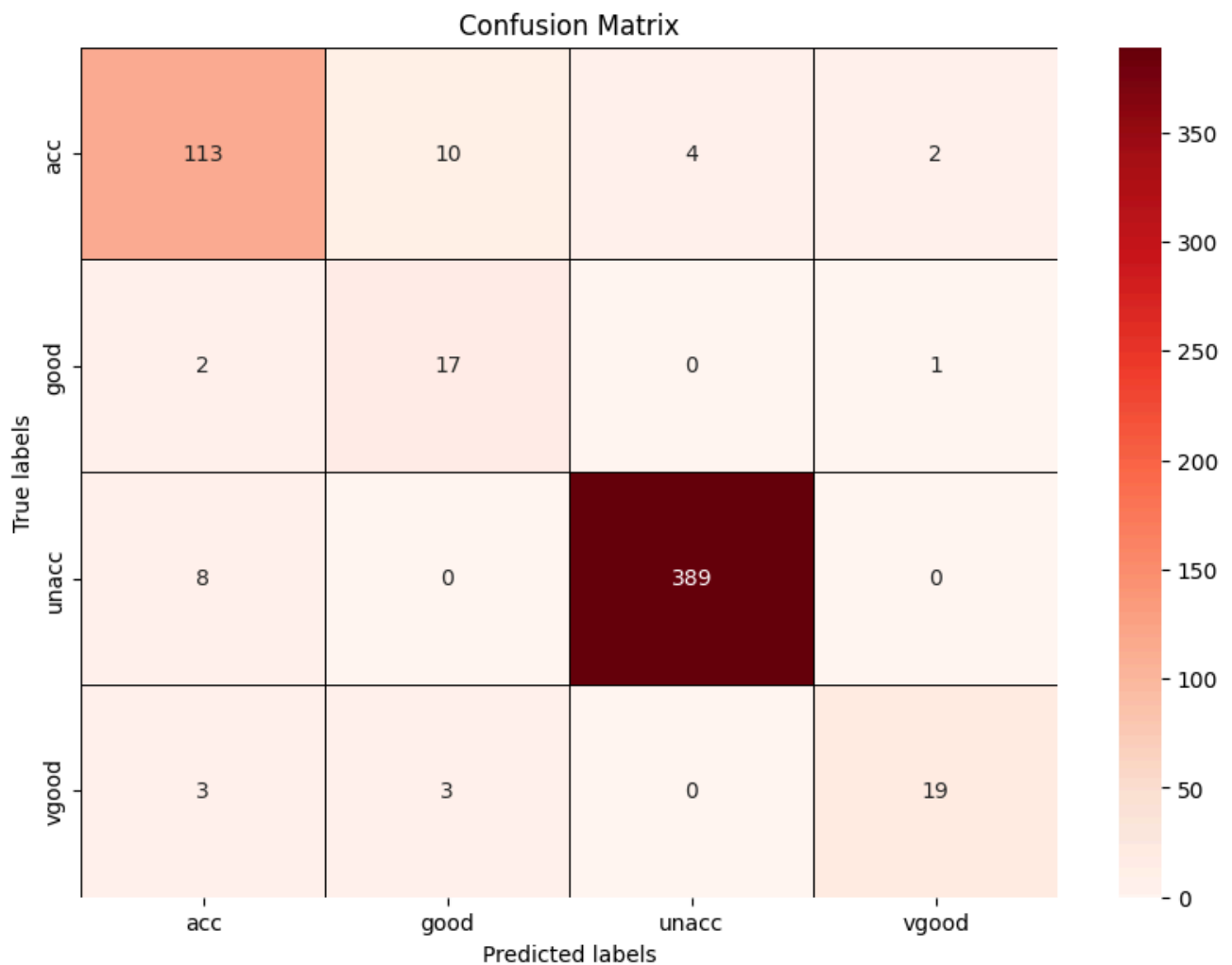
# Передбачення на тестових даних
y_pred_best = best_rf_model.predict(X_test)

# Обчислення матриці
cm = confusion_matrix(y_test, y_pred_best, labels=best_rf_model.classes_)


# Візуалізація матриці з новими кольорами
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds',
            xticklabels=best_rf_model.classes_,
            yticklabels=best_rf_model.classes_,
            linewidths=0.5, linecolor='black')

plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

```



```
cr = classification_report(y_test, y_pred_best, target_names=best_rf_model.classes_)
print(cr)
```



	precision	recall	f1-score	support
acc	0.90	0.88	0.89	129
good	0.57	0.85	0.68	20
unacc	0.99	0.98	0.98	397
vgood	0.86	0.76	0.81	25
accuracy			0.94	571
macro avg	0.83	0.87	0.84	571
weighted avg	0.95	0.94	0.94	571

Точність (Precision): Найвища точність спостерігається у класу 'unacc' (0.99), що вказує на рідкісні помилки моделі під час класифікації цього класу. Найнижча точність зафіксована у класу 'good' (0.57), що свідчить про труднощі моделі з правильним визначенням цього класу.

Повнота (Recall): Максимальна повнота у класу 'unacc' (0.98) вказує на те, що модель ефективно ідентифікує більшість випадків цього класу. Натомість, найнижча повнота зафіксована у класу 'vgood' (0.76), що свідчить про пропуски деяких прикладів цього класу.

F1-оцінка (F1-score): Найвища F1-оцінка спостерігається у класу 'unacc' (0.98), що свідчить про добру збалансованість між точністю та повнотою для цього класу. Найнижча F1-оцінка зафіксована у класу 'good' (0.68), що вказує на недостатню збалансованість цих двох показників для цього класу.

Загальна точність (Accuracy): Загальна точність складає 0.94, що означає, що модель правильно класифікує 94% усіх прикладів у тестовій вибірці.

Середні оцінки:

- Macro average: 0.83 для точності, 0.87 для повноти та 0.84 для F1-оцінки, що є середніми значеннями для всіх класів.
- Weighted average: 0.95 для точності, 0.94 для повноти та 0.94 для F1-оцінки, які враховують дисбаланс класів у тестовій вибірці.

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
```

```
# Отримуємо ймовірності класів для тестових даних
y_prob_best = best_rf_model.predict_proba(X_test)
```

```
# Ініціалізуємо словники для FPR, TPR та AUC
fpr = {}
tpr = {}
roc_auc = {}
```

```
# Обчислюємо FPR, TPR та AUC для кожного класу
for i, class_label in enumerate(best_rf_model.classes_):
    fpr[i], tpr[i], _ = roc_curve(y_test, y_prob_best[:, i], pos_label=class_label)
    roc_auc[i] = auc(fpr[i], tpr[i])
```

```
# Налаштовуємо кольори для кожного класу
```

```

# Налаштовуємо кольори для кожного класу
colors = ['#FF5733', '#33FF57', '#3357FF', '#FF33A8'] # Задаємо нові кольори

plt.figure(figsize=(12, 8))

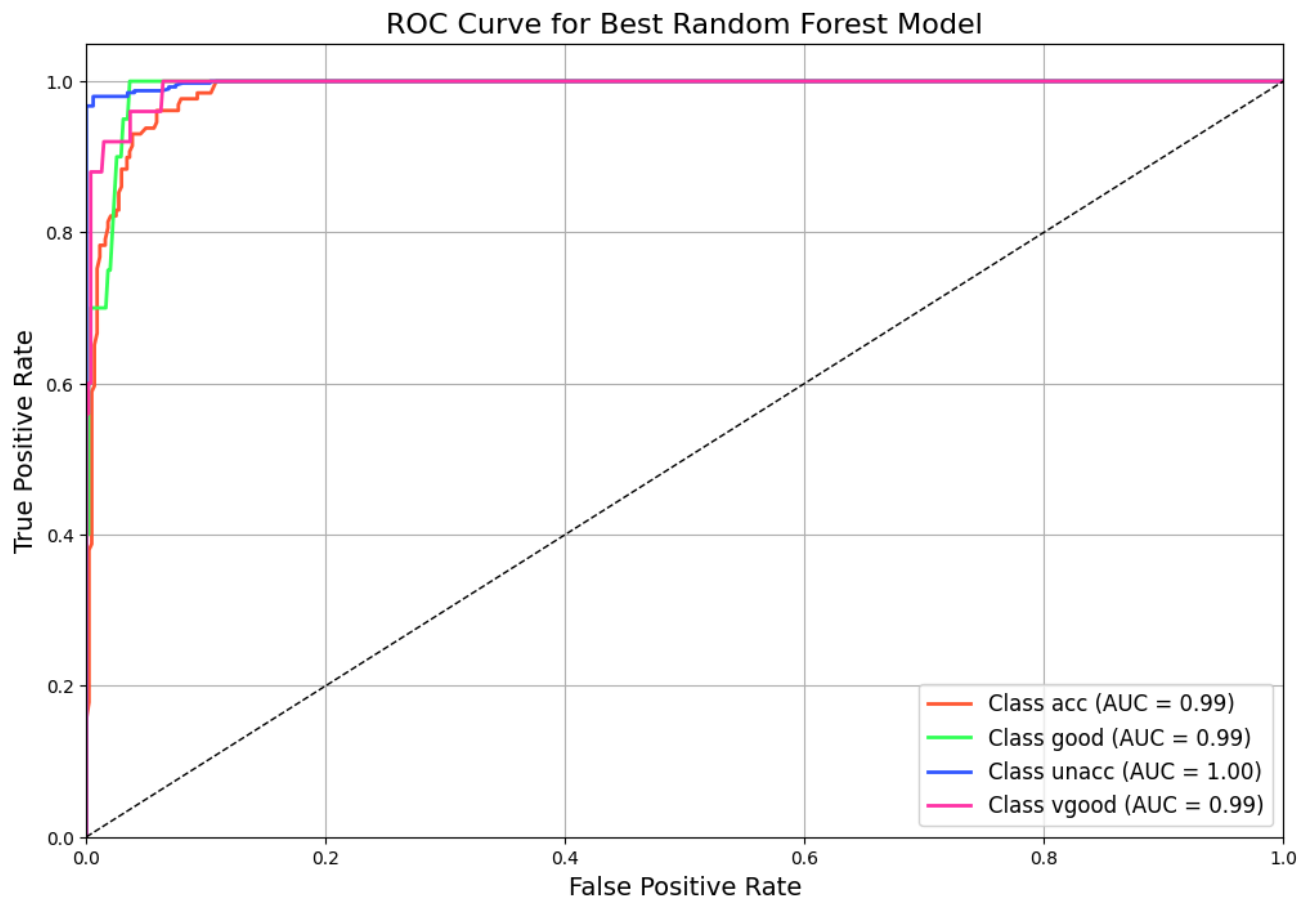
# Побудова графіків ROC для кожного класу
for i, class_label in enumerate(best_rf_model.classes_):
    plt.plot(fpr[i], tpr[i], color=colors[i], label=f'Class {class_label} (AUC = {roc_auc[i]:.2}

# Додаємо лінію випадкових прогнозів
plt.plot([0, 1], [0, 1], 'k--', linewidth=1)

# Налаштовуємо осі та заголовок
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.title('ROC Curve for Best Random Forest Model', fontsize=16)

# Додаємо легенду
plt.legend(loc='lower right', fontsize=12)
plt.grid(True) # Додаємо сітку для кращої читабельності
plt.show()

```



```
y_prob_best = best_rf_model.predict_proba(X_test)
auc = roc_auc_score(y_test, y_prob_best, multi_class='ovr')
print(f'AUC: {auc}')
```

➡ AUC: 0.993079306311704

Значення 0.9931 трохи перевищує попереднє 0.9926, що вказує на незначне поліпшення в здатності моделі розрізняти між класами після оптимізації гіперпараметрів.

✓ XGboost

```
import xgboost as xgb
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
```

Перетворюю категоріальні змінні у числові значення

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

Створюю екземпляр XGBClassifier та навчаємо модель на тренувальних даних

```
xgb_model = xgb.XGBClassifier(random_state=42)
xgb_model.fit(X_train, y_train_encoded)
```

➡

▼ XGBClassifier ⓘ

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, device=None, early_stopping_rounds=None,
               enable_categorical=False, eval_metric=None, feature_types=None,
               gamma=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=None, max_bin=None,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=None, max_leaves=None,
               min_child_weight=None, missing=nan, monotone_constraints=None,
               multi_strategy=None, n_estimators=None, n_jobs=None,
               num_parallel_tree=None, objective='multi:softprob', ...)
```

Робимо прогнозування на тестових даних

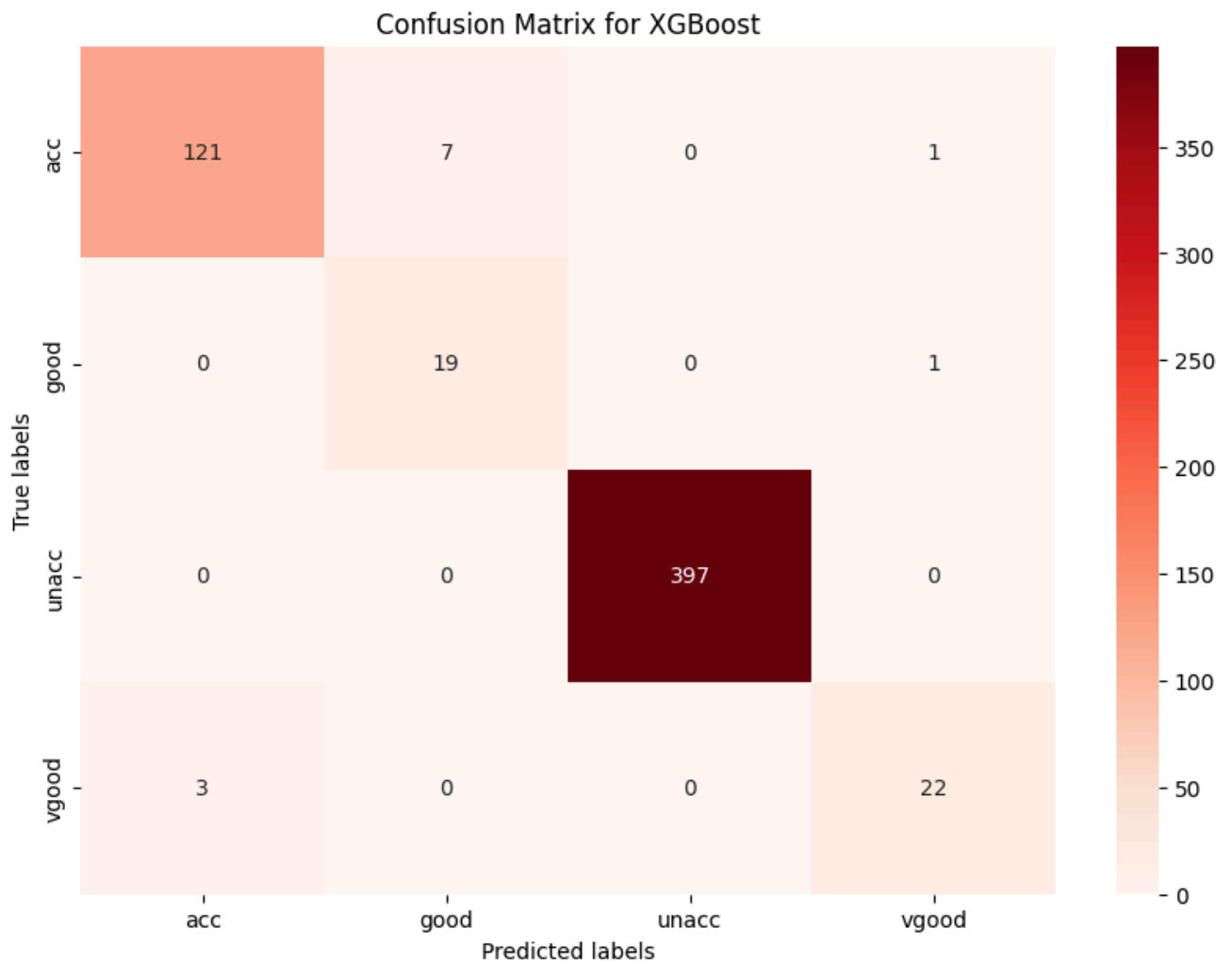
```
y_pred_encoded = xgb_model.predict(X_test)
y_pred = label_encoder.inverse_transform(y_pred_encoded)
```

```
# Зворотне перетворення передбачених міток
y_pred_encoded = xgb_model.predict(X_test)
y_pred = label_encoder.inverse_transform(y_pred_encoded)

# Обчислення матриці
cm_xgb = confusion_matrix(y_test, y_pred, labels=label_encoder.classes_)

# Візуалізація матриці
plt.figure(figsize=(10, 7))
sns.heatmap(cm_xgb, annot=True, fmt='d', cmap='Reds',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)

plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix for XGBoost')
plt.show()
```



```
cr_xgb = classification_report(y_test, y_pred, target_names=label_encoder.classes_)
print(cr_xgb)
```



```
precision    recall  f1-score   support

acc         0.98     0.94     0.96         129
```

good	0.73	0.95	0.83	20
unacc	1.00	1.00	1.00	397
vgood	0.92	0.88	0.90	25
accuracy			0.98	571
macro avg	0.91	0.94	0.92	571
weighted avg	0.98	0.98	0.98	571

✓ Загальна точність:

Модель **XGBClassifier** досягла вражаючої загальної точності 98% на тестовому наборі даних.

Класифікація за класами:

- **acc**: Модель демонструє відмінні результати з точністю 98% і відновлювальністю 94% для цього класу.
- **good**: Точність класифікації для цього класу становить 73%, проте відновлювальність залишається високою на рівні 95%. Це свідчить про те, що модель рідше помиляється при класифікації класу "good", але точність класифікації залишається нижчою.
- **unacc**: Модель ідеально класифікує цей клас, досягаючи 100% точності та відновлювальності.
- **vgood**: Для цього класу модель демонструє високі показники, з точністю 92% та відновлювальністю 88%.

Мета-метрики:

- **Макро-середнє**: Значення точності (91%) та F1-скорю (92%) є досить високими, що свідчить про загальну збалансованість моделі.
- **Вагове середнє**: Високі показники точності, відновлювальності та F1-скорю (98%) обумовлені переважно точною класифікацією класу "unacc", який має найбільшу кількість зразків.

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Отримуємо ймовірності класів для тестових даних
y_prob = xgb_model.predict_proba(X_test)

# Ініціалізуємо словники для FPR, TPR та AUC
fpr = {}
tpr = {}
roc_auc = {}

# Кількість класів
n_classes = len(label_encoder.classes_)

# Обчислюємо FPR, TPR та AUC для кожного класу
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_encoded == i, y_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```



```

# Налаштовуємо кольори для кожного класу
colors = ['#FF5733', '#33FF57', '#3357FF', '#FF33A8'] # Вибір нових кольорів

plt.figure(figsize=(12, 8))

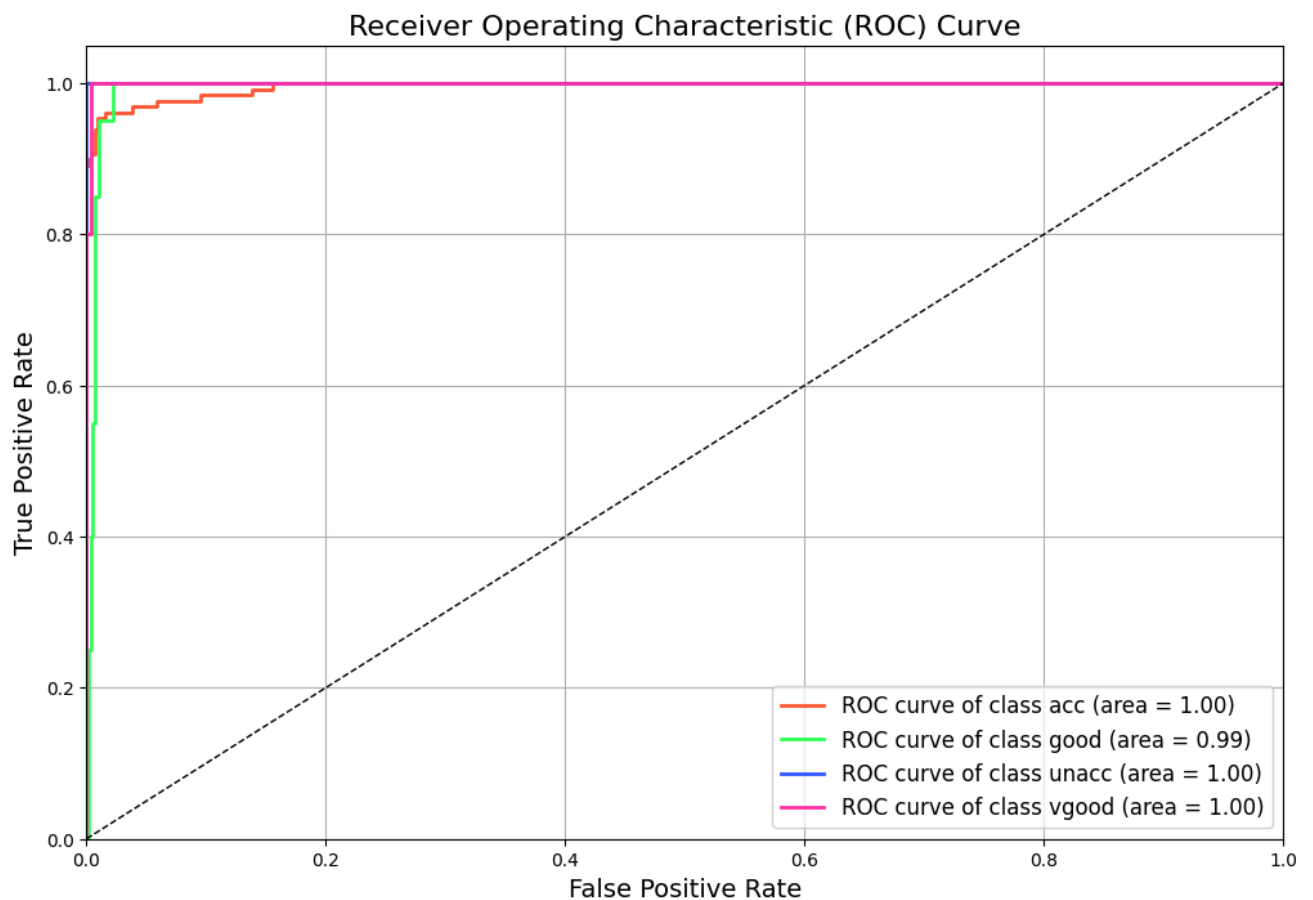
# Побудова графіків ROC для кожного класу
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], color=colors[i], label=f'ROC curve of class {label_encoder.classes

# Додаємо лінію випадкових прогнозів
plt.plot([0, 1], [0, 1], 'k--', linewidth=1)

# Налаштовуємо осі та заголовок
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize=14)
plt.ylabel('True Positive Rate', fontsize=14)
plt.title('Receiver Operating Characteristic (ROC) Curve', fontsize=16)

# Додаємо легенду
plt.legend(loc='lower right', fontsize=12)
plt.grid(True) # Додаємо сітку для кращої читабельності
plt.show()

```



```

y_prob_xgb = xgb_model.predict_proba(X_test)
auc_xgb = roc_auc_score(y_test_encoded, y_prob_xgb, multi_class='ovr')
print(f'AUC for XGBoost: {auc_xgb}')

```

➦ AUC for XGBoost: 0.9971999779136288

AUC для XGBoost: 0.9971999779136288

Цей показник є дуже високим, що свідчить про відмінну продуктивність моделі. Значення AUC, близьке до 1, вказує на те, що модель має високу здатність правильно класифікувати зразки між різними класами.

Покращення результатів: Модель XGBoost показала значне поліпшення в порівнянні з попередніми моделями, оскільки AUC зросла з 0.9926 до 0.9972. Це свідчить про те, що XGBoost краще справляється з класифікацією і має менше помилок, ніж інші моделі.

Загалом, XGBoost продемонструвала себе як дуже потужний інструмент для цієї задачі, з надзвичайно високим AUC, що робить її однією з найкращих моделей для класифікації в даному випадку.

