Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №1
Технології машинного навчання
Тема: «Метрики якості задач класифікації»

Виконав:
Студент групи ІА-12
Оверчук Дмитро Максимович

Перевірив:
Коломоєць С.О.

Київ 2024

Мета роботи — отримати знання основних метрик якості бінарної класифікації і варіантів тонкого налаштування алгоритмів класифікації.

Завдання на лабораторну роботу

1) Завантажте файл bioresponse.csv з папки “Лаб 1” за посиланням https://drive.google.com/drive/folders/1SYLvcaJ5XZQL2socPuGc6466AoEsrLue?usp=sharing

2) Навчіть 4 класифікатора, щоб передбачити поле "Activity" (біологічна відповідь молекули) з набору даних "bioresponse.csv":

- дрібне дерево рішень;

- глибоке дерево рішень;

- випадковий ліс на дрібних деревах;

- випадковий ліс на глибоких деревах.

3) Розрахуйте наступні метрики, щоб перевірити якість ваших моделей:

- частка правильних відповідей (accuracy);

- точність;

- повнота;

- $F1$-score;

- log-loss.

4) Побудуйте precision-recall і ROC-криві для ваших моделей.

5) Навчіть класифікатор, який уникає помилок II роду і розрахуйте для нього метрики якості.

Хід роботи

1. Завантаження та підготовка даних

Дані завантажуються з файлу bioresponse.csv. Виконується розділення на ознаки (X) та цільову змінну (y), а також ділення на навчальну та тестову вибірки:

```
def main():
    data = pd.read_csv('bioresponse.csv')

    # Розділення набору даних на ознаки (X) та цільову змінну (y).
    x = data.iloc[:, 1:]
    y = data.Activity

    # random_state фіксує випадковість під час навчання моделі, щоб результати
були відтворюваними між різними запусками
    random_state = 20

    # Розділення даних на тренувальний і тестовий набори.
    x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                        test_size=0.2,
                                                        random_state=random_state
                                                        )
```

- **test_size=0.2** – 20% даних використовуються для тестування, а 80% для навчання.
- **random_state=20** – фіксує випадковість для відтворюваності результатів.

2. Ініціалізація моделей

Створюються 4 моделі та додаткова модель для уникнення помилок II роду:

```
# Ініціалізація класифікаторів
shallow_tree = DecisionTreeClassifier(max_depth=3,
                                      random_state=random_state
                                      )
deep_tree = DecisionTreeClassifier(max_depth=15,
                                   random_state=random_state
                                   )
shallow_forest = RandomForestClassifier(n_estimators=100,
                                        max_depth=3,
                                        random_state=random_state
                                        )
deep_forest = RandomForestClassifier(n_estimators=100,
                                     max_depth=15,
                                     random_state=random_state
                                     )
avoid_type_2_error_classifier = DecisionTreeClassifier(
    max_depth=5,
    class_weight={0: 1, 1: 5},
```

```
    random_state=random_state
)
```

- **max_depth** – обмежує глибину дерев рішень для контролю складності моделей.
- **n_estimators** – кількість дерев у випадковому лісі.
- **class_weight={0: 1, 1: 5}** – підвищує вагу класу "1" для моделі, що уникає помилок II роду.

3. Навчання, оцінка та візуалізація моделей

Функція train_eval_visualize_model приймає модель, її назву та дані для навчання й тестування. Вона навчає модель, обчислює метрики та будує графіки Precision-Recall та ROC-криві.

```
# Навчання, оцінка та візуалізація роботи моделей
train_eval_visualize_model(shallow_tree, "Shallow Tree", x_train, y_train, x_test,
y_test)
train_eval_visualize_model(deep_tree, "Deep Tree", x_train, y_train, x_test,
y_test)
train_eval_visualize_model(shallow_forest, "Shallow Forest", x_train, y_train,
x_test, y_test)
train_eval_visualize_model(deep_forest, "Deep Forest", x_train, y_train, x_test,
y_test)
train_eval_visualize_model(avoid_type_2_error_classifier, "Avoid Type II Error
Tree", x_train, y_train, x_test,
                           y_test)
```

```
# Допоміжна функція для навчання, оцінки та візуалізації роботи моделі.
def train_eval_visualize_model(model, model_name, x_train, y_train, x_test,
y_test):
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    y_pred_prob = model.predict_proba(x_test)
    y_pred_class_of_interest_probabilities = y_pred_prob[:, 1]

    print(f"-----{model_name}-----")
    print_metrics_to_terminal(y_pred, y_pred_class_of_interest_probabilities,
y_test.values)
    plot_precision_recall(model_name, y_pred_class_of_interest_probabilities,
y_test.values)
    plot_roc(model_name, y_pred_class_of_interest_probabilities, y_test.values)
    print("\n")
```

4. Розрахунок метрик (файл metrics_calc.py)

Функції для обчислення метрик:

- **calc_accuracy** – обчислює частку правильних відповідей.

- **calc_precision** – оцінює точність моделі для позитивних класів.
- **calc_recall** – вимірює повноту, враховуючи пропущені позитивні значення.
- **calc_f1_score** – гармонійне середнє між точністю і повнотою.
- **calc_log_loss** – обчислює логарифмічні втрати моделі.

```
# Функції для обчислення метрик
def calc_accuracy(pred_values, true_values):
    number_of_predictions = len(pred_values)
    correct_predictions_count = 0
    for i in range(len(pred_values)):
        if pred_values[i] == true_values[i]:  # Перевіряємо чи прогноз
відповідаємо істинному значенню
            correct_predictions_count += 1
    return correct_predictions_count / number_of_predictions  #
Розрахуємо частку правильних відповідей


def calc_precision(pred_values, true_values):
    true_positives_count = 0
    false_positives_count = 0
    for i in range(len(pred_values)):
        if pred_values[i] == 1 and true_values[i] == 1:  # True
positive
            true_positives_count += 1
        elif pred_values[i] == 1 and true_values[i] == 0:  # False
positive
            false_positives_count += 1
    return true_positives_count / (true_positives_count +
false_positives_count)  # Обчислення точності моделі


def calc_recall(pred_values, true_values):
    true_positives_count = 0
    false_negatives_count = 0
    for i in range(len(pred_values)):
        if pred_values[i] == 1 and true_values[i] == 1:  # True
positive
            true_positives_count += 1
        elif pred_values[i] == 0 and true_values[i] == 1:  # False
negative
            false_negatives_count += 1
    return true_positives_count / (true_positives_count +
false_negatives_count)  # Обчислення повноти моделі


def calc_f1_score(pred_values, true_values):
    precision = calc_precision(pred_values, true_values)
    recall = calc_recall(pred_values, true_values)
    return 2 * precision * recall / (precision + recall)  # Обчислення
F1-score моделі


def calc_log_loss(pred_values_prob, true_values):
    all_values_count = len(pred_values_prob)
    log_sum = 0
```

```
    for i in range(len(pred_values_prob)):
        predicted_probability = pred_values_prob[i]
        real = true_values[i]
        # Додано невелику костанту для уникнення обчилення логарифму
від 0
        addition = real * math.log(predicted_probability + 1e-9) + \
                   (1 - real) * math.log(1 - predicted_probability +
1e-9)
        log_sum += addition

    return -1 / all_values_count * log_sum  # Обчислення log-loss
моделі
```

5. *Побудова Precision-Recall та ROC-кривих (файл plotting.py)*

Функції для побудови графіків:

- **plot_roc** – будує ROC-криву та обчислює AUC (площу під кривою).
- **plot_precision_recall** – будує криву Precision-Recall для моделі.

```
# Функції для побудови кривих
def plot_roc(model_name, pred_values_prob, true_values):
    roc_auc = roc_auc_score(true_values, pred_values_prob)
    false_positive_rates, true_positive_rates, thresholds =
roc_curve(true_values, pred_values_prob)
    plt.plot(false_positive_rates, true_positive_rates, marker='.',
label=f'ROC AUC = {roc_auc:.2f}')
    plt.xlabel("False positive rate")
    plt.ylabel("True positive rate")
    plt.title(f"{model_name} ROC Curve")
    plt.legend()
    plt.show()


def plot_precision_recall(model_name, predicted_probability,
true_values):
    precision, recall, thresholds = precision_recall_curve(true_values,
predicted_probability)
    plt.plot(thresholds, precision[1:], color='blue',
label='Precision')
    plt.plot(thresholds, recall[1:], color='orange', label='Recall')
    plt.xlabel("Threshold")
    plt.ylabel("Score")
    plt.title(f"{model_name} Precision-Recall Curve")
    plt.legend()
    plt.show()
```

Висновки: У ході роботи було завантажено набір даних, проведено розділення на навчальний і тестовий набори, навчено 5 класифікаторів, зокрема дерево рішень та випадковий ліс різної глибини. Оцінено якість моделей за метриками accuracy, precision, recall, F1-score та log-loss. Побудовано ROC та Precision-Recall криві для візуального аналізу. Модель із врахуванням ваг класів

показала підвищену повноту, що зменшило кількість помилок II роду, демонструючи ефективність налаштувань class_weight.