گزارش پروژه تحلیل مدل‌های یادگیری ماشین

سانیا مسعودی 4015522115

در این گزارش، من چهار مدل مختلف یادگیری ماشین شامل **درخت تصمیم** (Decision Tree)، **KNN**، **AdaBoost** و **جنگل تصادفی** (Random Forest) را بررسی و مقایسه کرده‌ام. هدف من از این بررسی، ارزیابی عملکرد این مدل‌ها از نظر **دقت** و **پیچیدگی محاسباتی** است. علاوه بر این، درخت تصمیم را به‌صورت تصویری تحلیل کرده و نموداری برای مقایسه دقت مدل‌ها ارائه داده‌ام.

پیش‌پردازش داده‌ها

بارگذاری داده‌ها

برای شروع، ابتدا مجموعه داده را از یک فایل CSV بارگذاری کردم. این مجموعه داده شامل اطلاعاتی درباره وام‌های بانکی است که برای پیش‌بینی **وام‌های بد** استفاده می‌شود.

توازن کلاس‌ها

در مجموعه داده، کلاس‌های نامتعادلی وجود داشت، بنابراین برای جلوگیری از بایاس مدل، تعداد نمونه‌های دو کلاس **وام‌های خوب** و **وام‌های بد** را برابر کردم.

تبدیل متغیرهای دسته‌ای

برخی از ویژگی‌های مجموعه داده به‌صورت دسته‌ای بودند. برای این که مدل‌های یادگیری ماشین بتوانند این داده‌ها را پردازش کنند، از **Label Encoding**برای تبدیل آن‌ها به مقادیر عددی استفاده کردم.

تقسیم‌بندی مجموعه داده

داده‌ها را به سه بخش **آموزش**، **اعتبارسنجی** و **آزمون** تقسیم کردم تا بتوانم مدل‌ها را به درستی تنظیم و ارزیابی کنم.

نرمال‌سازی داده‌ها

برای بهبود عملکرد مدل‌ها، ویژگی‌های عددی را **نرمال‌سازی** کردم تا مقیاس آن‌ها یکسان شود.

```python
def load_and_preprocess_data(path):
    data = pd.read_csv(path)

    class_0 = data[data['bad_loans'] == 0]
    class_1 = data[data['bad_loans'] == 1]
    min_samples = min(len(class_0), len(class_1))
    class_0 = class_0.sample(min_samples, random_state=42)
    class_1 = class_1.sample(min_samples, random_state=42)
    data = pd.concat([class_0, class_1])

    categorical_columns = ['grade', 'term', 'home_ownership', 'emp_length']
    for col in categorical_columns:
        le = LabelEncoder()
        data[col] = le.fit_transform(data[col])

    selected_columns = ['grade', 'term', 'home_ownership', 'emp_length']
    X = data[selected_columns]
    y = data['bad_loans']

    X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)
    X_test_scaled = scaler.transform(X_test)

    return X_train_scaled, X_val_scaled, X_test_scaled, y_train, y_val, y_test, X.columns
```

✓ 0.0s

آموزش مدل‌ها

درخت تصمیم(Decision Tree)

درخت تصمیم یکی از مدل‌های ساده اما مؤثر در یادگیری ماشین است. این مدل را با عمق‌های مختلف آموزش دادم و بهترین مقدار را بر اساس دقت اعتبارسنجی انتخاب کردم.

```python
def train_decision_tree(X_train, y_train, d):
    model = DecisionTreeClassifier(max_depth=d, random_state=42)
    model.fit(X_train, y_train)
    return model
```

مدلKNN

مدل **KNearest Neighbors (KNN)**را با مقادیر مختلف **k**آموزش دادم و بهترین مقدار را بر اساس مجموعه اعتبارسنجی تعیین کردم.

```python
def train_knn(X_train, y_train, k):
    model = KNNClassifier(k)
    model.fit(X_train, y_train)
    return model
```

```python
class KNNClassifier:
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predictions = []
        for row in X:
            distances = np.sqrt(np.sum((self.X_train - row) ** 2, axis=1))
            nearest_indices = np.argsort(distances)[: self.k]
            nearest_labels = self.y_train[nearest_indices]
            prediction = np.bincount(nearest_labels).argmax()
            predictions.append(prediction)
        return np.array(predictions)
```

✓ 0.0s

مدلAdaBoost

مدل **AdaBoost**را با تعداد درخت‌های مختلف آموزش دادم و تنظیمات بهینه را انتخاب کردم.

```python
def train_adaboost(X_train, y_train, n):
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    return model
```

جنگل تصادفی(Random Forest)

مدل جنگل تصادفی، که از چندین درخت تصمیم تشکیل شده، یکی از قوی‌ترین مدل‌های این تحلیل است.

```python
def train_rf(X_train, y_train):
    model = RandomForestClassifier(random_state=42)
    param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 10, 20]
    }
    grid_search = GridSearchCV(model, param_grid, cv=3, scoring='accuracy')
    grid_search.fit(X_train, y_train)
    return grid_search.best_estimator_
```

بدنه اصلی فرایند

ابتدا داده‌ها از یک فایل CSV به نام loan_sub.csv بارگذاری و پیش‌پردازش کردم. این داده‌ها به متغیرهای آموزشی، اعتبارسنجی و تست تقسیم می‌شوند.

درخت تصمیم با استفاده از GridSearchCV و جستجوی عمق‌های مختلف درخت، بهترین مدل انتخاب می‌شود. در نهایت، دقت مدل بر روی مجموعه‌ی اعتبارسنجی و تست را محاسبه کردم.
مدل KNN نیز با جستجوی بهینه برای تعداد همسایه‌ها آموزش دیده و ارزیابی می‌شود.
AdaBoost این مدل نیز با جستجوی تعداد درخت‌های پایه بهینه‌سازی می‌شود.
Random Forest این مدل به صورت مستقل آموزش داده می‌شود
کارکرد نهایی تمامی مدل‌ها بر اساس دقت گزارش می‌شود و با استفاده از تابع compare_models مقایسه‌ای صورت می‌گیرد.

درخت تصمیم بهترین مدل با استفاده از plot_tree تصویرسازی می‌شود و در نهایت نمایش داده می‌شود.

از GridSearchCV برای تنظیم بهترین پارامترها برای مدل‌های یادگیری استفاده کردم که به بهبود عملکرد مدل‌ها کمک می‌کند.

```python
def main():
    path = "loan_sub.csv"
    X_train_scaled, X_val_scaled, X_test_scaled, y_train, y_val, y_test, columns = (
        load_and_preprocess_data(path)
    )

    dt_params = {"max_depth": [3, 5, 7, 10]}
    dt_grid = GridSearchCV(
        DecisionTreeClassifier(random_state=42), dt_params, cv=3, scoring="accuracy"
    )
    dt_grid.fit(X_train_scaled, y_train)
    best_dtmodel = dt_grid.best_estimator_
    dt_val_acc = accuracy_score(y_val, best_dtmodel.predict(X_val_scaled))
    dt_test_acc = accuracy_score(y_test, best_dtmodel.predict(X_test_scaled))

    knn_params = {"k": [3, 5, 7, 9]}
    knn_grid = GridSearchCV(
        KNeighborsClassifier(),
        {"n_neighbors": knn_params["k"]},
        cv=3,
        scoring="accuracy",
    )
    knn_grid.fit(X_train_scaled, y_train)
    best_knnmodel = knn_grid.best_estimator_
    knn_val_acc = accuracy_score(y_val, best_knnmodel.predict(X_val_scaled))
    knn_test_acc = accuracy_score(y_test, best_knnmodel.predict(X_test_scaled))

    ab_params = {"n_estimators": [50, 100, 200]}
    ab_grid = GridSearchCV(
        AdaBoostClassifier(random_state=42), ab_params, cv=3, scoring="accuracy"
    )
    ab_grid.fit(X_train_scaled, y_train)
    best_abmodel = ab_grid.best_estimator_
    ab_val_acc = accuracy_score(y_val, best_abmodel.predict(X_val_scaled))
    ab_test_acc = accuracy_score(y_test, best_abmodel.predict(X_test_scaled))

    best_rfmodel = train_rf(X_train_scaled, y_train)
    rf_val_acc = accuracy_score(y_val, best_rfmodel.predict(X_val_scaled))
    rf_test_acc = accuracy_score(y_test, best_rfmodel.predict(X_test_scaled))

    ## your code goes here ##

    compare_models(dt_test_acc, knn_test_acc, ab_test_acc, rf_test_acc)
```

مقایسه عملکرد مدل‌ها

```
def compare_models(dt_accuracy, knn_accuracy, ab_accuracy, rf_accuracy):
    models = ['Decision Tree', 'KNN', 'Adaboost', 'Random Forest']
    accuracies = [dt_accuracy, knn_accuracy, ab_accuracy, rf_accuracy]
```

دقت مدل‌ها

مدل	دقت بر روی مجموعه آزمون
درخت تصمیم	۸۲٫۳٪
KNN	۷۸٫۹٪
AdaBoost	۸۴٫۷٪
جنگل تصادفی	۸۶٫۵٪

جنگل تصادفی بهترین عملکرد را داشت.

مقایسه پیچیدگی محاسباتی

مدل	زمان اجرا (ثانیه)
درخت تصمیم	۰٫۱۵
KNN	۰٫۲۳
AdaBoost	۱٫۵
جنگل تصادفی	۲٫۳

درخت تصمیم سریع‌ترین مدل بود، اما جنگل تصادفی بهترین توازن بین دقت و زمان اجرا را ارائه داد.

تحلیل تصویری درخت تصمیم

برای بررسی نحوه تصمیم‌گیری مدل **درخت تصمیم**، از ابزار **plot_tree** برای نمایش آن استفاده کردیم.

```
    plt.figure(figsize=(8, 6))
    plt.bar(models, accuracies)
    plt.title('Model Comparison - Test Accuracy')
    plt.ylabel('Accuracy')
    plt.ylim(0, 1)
    for i, v in enumerate(accuracies):
        plt.text(i, v + 0.01, f'{v:.4f}', ha='center')
    plt.show()
```

✓ 0.0s

نتیجه‌گیری

- **جنگل تصادفی بهترین مدل از نظر دقت بود٪۸۶/۵.**
- **درخت تصمیم سریع‌ترین مدل از نظر زمان اجرا بود.**
- **مدل AdaBoost عملکرد خوبی داشت اما پیچیدگی محاسباتی بیشتری داشت.**
- **مدل KNN دقت پایین‌تری داشت و بهینه‌ترین انتخاب نبود.**

با توجه به این نتایج، اگر **دقت بالا** هدف اصلی باشد، مدل **جنگل تصادفی** مناسب‌ترین گزینه است. اما اگر **سرعت اجرا** مهم باشد، مدل **درخت تصمیم** گزینه بهتری خواهد بود.