Исследование композиций алгоритмов классификации

Кирилл Грошев (475 группа)

1 Постановка задачи

Рассматривается задача классификации ghouls and ghosts. Предлагается исследовать эффективность различных алгоритмов классификации и различные их композиции.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import preprocessing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier, VotingC
```

In [2]:
```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

In [3]:
```
train.head()
```

Out[3]:

	id	bone_length	rotting_flesh	hair_length	has_soul	color	type
0	0	0.354512	0.350839	0.465761	0.781142	clear	Ghoul
1	1	0.575560	0.425868	0.531401	0.439899	green	Goblin
2	2	0.467875	0.354330	0.811616	0.791225	black	Ghoul
3	4	0.776652	0.508723	0.636766	0.884464	black	Ghoul
4	5	0.566117	0.875862	0.418594	0.636438	green	Ghost

In [4]:
```
train = train.drop("id", axis=1)
test = test.drop("id", axis=1)
```

In [5]:
```
train.head()
```

Out[5]:

	bone_length	rotting_flesh	hair_length	has_soul	color	type
0	0.354512	0.350839	0.465761	0.781142	clear	Ghoul
1	0.575560	0.425868	0.531401	0.439899	green	Goblin
2	0.467875	0.354330	0.811616	0.791225	black	Ghoul
3	0.776652	0.508723	0.636766	0.884464	black	Ghoul
4	0.566117	0.875862	0.418594	0.636438	green	Ghost

2 Базовое решение

Для начала рассмотрим сам датасет. Пропусков данных нет, сами данные сбалансированы. Признак color является категориальным, соответственно придется использовать labelencoder. Поскольку данные сбалансированы будем использовать в качестве метрики точность. В качестве базового решения можно например использовать решающее дерево. Подбор параметров пока производить не будем, так как в будущем будут использоваться более продвинутые алгоритмы, а цель решающего дерева - задать эталонную точность для оценки эффективности продвинутого решения.

```
In [6]: data = pd.get_dummies(train.drop('type', axis=1))
        X_train, X_test, y_train, y_test = train_test_split(data, train['type'], test_size=0.3,

        tree_cl = DecisionTreeClassifier(random_state=42)
        tree_cl.fit(X_train, y_train)
        y_pred = tree_cl.predict(X_test)

        print(accuracy_score(y_test, y_pred))
```

```
0.7142857142857143
```

3 Основное решение

Для начала исследуем поведение популярных алгоритмов для решения задачи классификации. Параметры будут подбираться с помощью кросс-валидации, разбиение на 5 блоков, в качестве метрики используется точность. Параметры по которым будет вестись поиск указаны в объекте grid. В качестве ответов получим набор параметров и точность. Заметим, что использование kNN нецелесообразно, так как результат ниже чем у baseline решения. Далее объеденим эти алгоритмы с оптимальными параметрами с помощью голосования по большинству.

```
In [7]: X_train = pd.get_dummies(train.drop('type', axis=1))
        y_train = train['type']
```

```
In [8]: lr = LogisticRegression(random_state=42)
        grid = {'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1, 2, 3, 5, 10]}
        clf = GridSearchCV(lr, param_grid = grid, scoring = 'accuracy', cv = 5)
        clf.fit(X_train, y_train)
        print(clf.best_params_)
        print(clf.best_score_)

        lr = LogisticRegression(random_state=42, penalty='l1', C=10)
```

```
{'C': 10, 'penalty': 'l1'}
0.7304582210242587
```

```
In [9]: knn = KNeighborsClassifier()
        grid = {'n_neighbors':[3, 5, 10, 20], 'leaf_size':[20, 30, 50], 'p':[1, 2], 'weights':['
```

```
        clf = GridSearchCV(knn, param_grid = grid, scoring = 'accuracy', cv = 5)
        clf.fit(X_train, y_train)
        print(clf.best_params_)
        print(clf.best_score_)

        knn = KNeighborsClassifier(leaf_size=20, n_neighbors=5, p=2, weights='uniform')
```

```
{'leaf_size': 20, 'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}
0.6684636118598383
```

```
In [10]: svm = SVC(random_state=42)
         grid = {'kernel':['linear', 'rbf'], 'C':[1, 3, 5, 10]}
         clf = GridSearchCV(svm, param_grid = grid, scoring = 'accuracy', cv = 5)
         clf.fit(X_train, y_train)
         print(clf.best_params_)
         print(clf.best_score_)

         svm = SVC(random_state=42, C=5, kernel='linear')
```

```
{'C': 5, 'kernel': 'linear'}
0.7250673854447439
```

```
In [11]: gb = GradientBoostingClassifier(random_state=42)
         grid = {'learning_rate':[0.05, 0.1, 0.5], 'n_estimators':[100, 200, 500], 'max_depth':[
         clf = GridSearchCV(gb, param_grid = grid, scoring = 'accuracy', cv = 5)
         clf.fit(X_train, y_train)
         print(clf.best_params_)
         print(clf.best_score_)

         gb = GradientBoostingClassifier(random_state=42, learning_rate=0.05, max_depth=2, n_est
```

```
{'learning_rate': 0.05, 'max_depth': 2, 'n_estimators': 100}
0.7331536388140162
```

```
In [12]: rf = RandomForestClassifier(random_state=42)
         grid = {'n_estimators':[10, 20, 50, 100], 'criterion':['gini', 'entropy'], 'max_depth':
         clf = GridSearchCV(rf, param_grid = grid, scoring = 'accuracy', cv = 5)
         clf.fit(X_train, y_train)
         print(clf.best_params_)
         print(clf.best_score_)

         rf = RandomForestClassifier(random_state=42, criterion='entropy', max_depth=5, n_estima
```

```
{'criterion': 'entropy', 'max_depth': 5, 'n_estimators': 10}
0.7115902964959568
```

4 Голосование

Нетрудно заметить, что объединение алгоритмов с помощью голосования дает прирост точности примерно 0.1. Более точных результатов можно достичь если провести дополнительный тюнинг параметров. Тем не менее, удалось получить улучшение относительно каждого из отдельных алгоритмов классификации.

In [14]:
```
voting = VotingClassifier(estimators=[('lr', lr), ('svc', svm), ('gbc', gb), ('rfc', rf
voting.fit(X_train, y_train)
voting.score(X_train, y_train)
```

Out[14]:
```
0.8274932614555256
```