«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ВЫСШАЯ ШКОЛА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНФОРМАЦИОННЫХ СИСТЕМ

ОТЧЕТ

Выполнил:
Студент: 3 курса
Группа: 11-406
Муллин Айнур Алмазович

Научный руководитель:
Новиков Петр Андреевич

Постановка задачи

Ссылка на задание
https://www.kaggle.com/c/ghouls-goblins-and-ghosts-boo

Набор данных, который предоставила компания Kaggle - это данные о монстрах. Такие, как: "Призраки", "Гоблины", "Вампиры".

У каждого монстра есть свои параметры:

1. ID - Идентификатор - число идентифицирующее данного существа
2. Bone_length - Длина костей - средняя длина костей, нормализованных от 0 до 1
3. Rotting_flesh - Гниющая плоть - в процентах
4. Hair_length- Длина волос - средняя длина волос, нормализованных от 0 до 1
5. Has_soul - Присутствие души - в процентах
6. Color - Цвет - доминирующий цвет этого существа ('white','black','clear','blue','green','blood')
7. Type - тип этого существа ('Ghost', 'Goblin', 'Ghoul')

Данные представлены в виде двух файлов test.csv и train.csv

Необходимо научиться максимально точно определять вид монстра по их параметрам.

План выполнение задания

Для выполнения задания были выделены такие шаги:

1. Необходимо научиться считывать данные из формата *.csv
2. Отобразить данные в виде графиков.
3. Преобразовать данные в нужный вид
4. Выделить список классификаторов, определить для них лучшие параметры.
5. Сделать проверку данных на этих классификаторах
6. Вывести результат работы в виде файла submission.csv
7. Написать вывод

Считывание данных из файлов

Первым шагом необходимо считать данные с файлов.

Для этого было использовано библиотека Pandas, с помощью которого можно без проблем считать данные формата .csv

```
import pandas as pd
train = pd.read_csv("./input/train.csv")
test = pd.read_csv("./input/test.csv")
```

Отрисовка данных в виде графиков

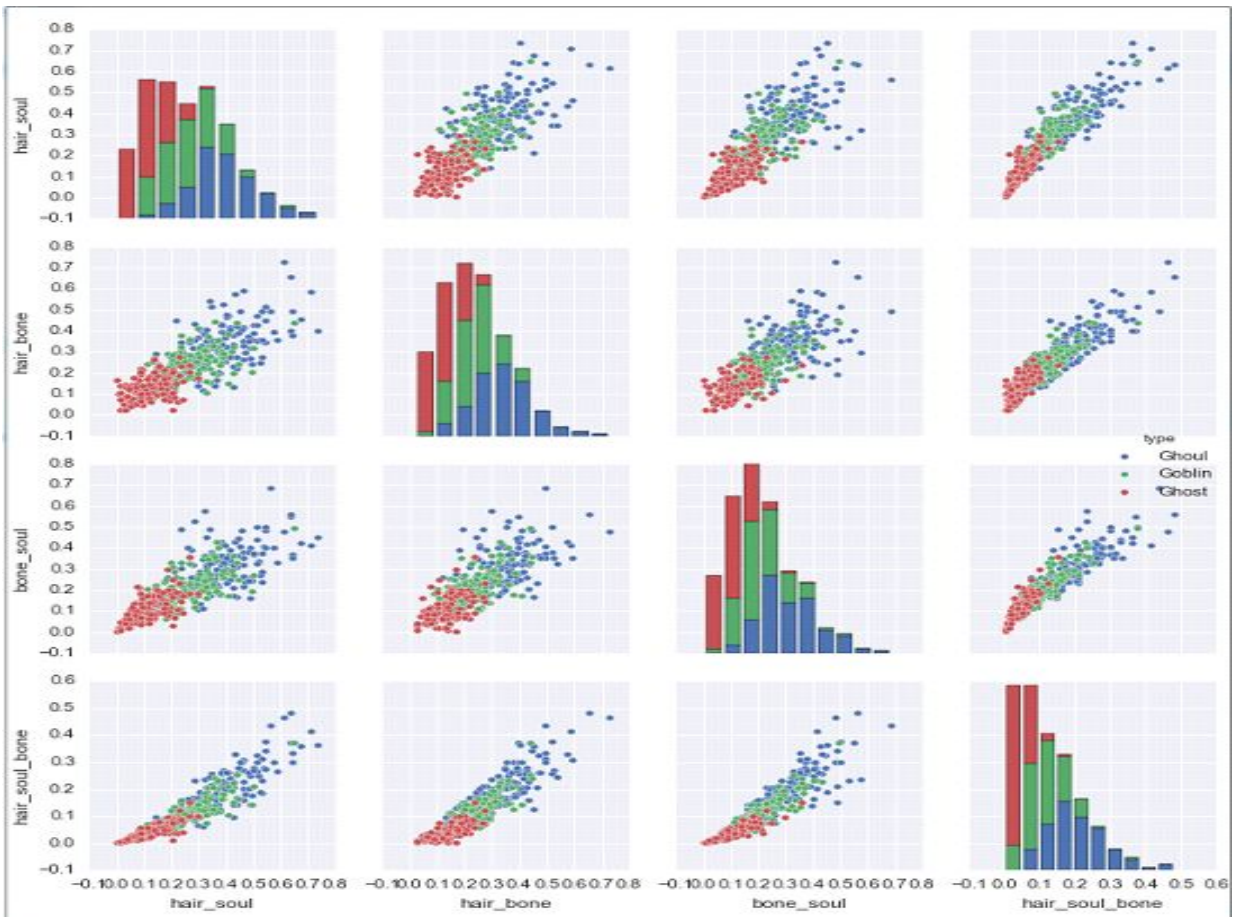
Был написан метод, который будет заниматься отображением данных.

```python
import matplotlib.pyplot as plt
import seaborn as sns

def show(trainset, testset):
    sns.set()
    trainset['hair_soul'] = trainset.apply(lambda row: row['hair_length'] * row['has_soul'], axis=1)
    trainset['hair_bone'] = trainset.apply(lambda row: row['hair_length'] * row['bone_length'], axis=1)
    trainset['bone_soul'] = trainset.apply(lambda row: row['bone_length'] * row['has_soul'], axis=1)
    trainset['hair_soul_bone'] = trainset.apply(lambda row: row['hair_length'] * row['has_soul'] * row['bone_length'],
                                                axis=1)

    testset['hair_soul'] = testset.apply(lambda row: row['hair_length'] * row['has_soul'], axis=1)
    testset['hair_bone'] = testset.apply(lambda row: row['hair_length'] * row['bone_length'], axis=1)
    testset['bone_soul'] = testset.apply(lambda row: row['bone_length'] * row['has_soul'], axis=1)
    testset['hair_soul_bone'] = testset.apply(lambda row: row['hair_length'] * row['has_soul'] * row['bone_length'],
                                              axis=1)

    sns.pairplot(trainset[["hair_soul", "hair_bone", "bone_soul", "hair_soul_bone", "type"]], hue="type")
    plt.show()
```



На вход этот метод получает соответственно train и test, полученные из файлов.
Пример вызова метода:

```
vizualization.show(trainset, testset)
```

На данном графике:
Красный цвет - это Призраки
Синий цвет - это Вампиры
Зеленый цвет - это Гоблины

Из графиков видно, что по всем параметрам Гоблины находятся между Призраками и Вампирами
Также, что Вампиры находятся выше чем Призраки

Работа над самими данными

Основные данные, на которых будут тренироваться алгоритмы, были разделены на две части X и y

```
train = pd.read_csv("./input/train.csv")
test = pd.read_csv("./input/test.csv")

y = train["type"]
X = train.drop(['color','id','type'],axis=1)
```

Разделение на тестовые и тренировочные данные происходит с помощью

```
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,
                                                    random_state=42)
```

Работа над классификаторами

Было найдены наиболее известные алгоритмы. Такие, как: RandomForest, DecisionTree, SVC, MLP и другие.

```
classifiers = [
    RandomForestClassifier(max_features='sqrt',
                           criterion='entropy',
                           n_estimators=10),
    AdaBoostClassifier(random_state=42,
                       n_estimators=500,
                       learning_rate=0.01),
    DecisionTreeClassifier(min_samples_split=30,
                           max_depth=5),
    GradientBoostingClassifier(subsample=0.8,
                               learning_rate=0.1,
                               n_estimators=500),
    SVC(kernel='rbf', C=0.02, probability=True),
    KNeighborsClassifier(n_neighbors=10),
    LinearDiscriminantAnalysis(),
    QuadraticDiscriminantAnalysis(),
    MLPClassifier(hidden_layer_sizes=(9,),
                  max_iter=1700,
                  alpha=0.10000000000000001,
                   solver='lbfgs')
]
```

Для каждого алгоритма подобраны параметры с помощью GridSearch

Пример проверки с помощью GridSearch для MLPClassifier

```
mlp = MLPClassifier(verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)

parameter_grid = {'solver': ['lbfgs'], 'max_iter': [1600],
                  'alpha': 10.0 ** -np.arange(1, 7),
                  'hidden_layer_sizes': np.arange(5, 15)}

grid_search = GridSearchCV(mlp, param_grid=parameter_grid,
                           scoring='accuracy',
                           cv=StratifiedKFold(5))
grid_search.fit(X_train, y_train)

print('Best score mlp: {}'.format(grid_search.best_score_))
print('Best parameters mlp: {}'.format(grid_search.best_params_))
```

Для этого алгоритма самыми лучшими параметрами оказались
hidden_layer_sizes=9
max_iter=1700
alpha=0.10000000000000001
solver='lbfgs'

Таким же способом были проверены и остальные алгоритмы.

Проверка классификаторов на данных

После того, как все классификаторы и данные готовы, можно приступить к поиску наилучшего по точности и наилучшего по времени алгоритма.
Для этого был написан код, который представлен ниже и закомментирован.

```
columns = ['Classifier', 'Accuracy', 'Time Learn', 'Time Predict'] #Объявление колонок таблицы
result = pd.DataFrame(columns=columns) #Объявление массива с колонками

for classifier in classifiers: #Прохождение по всем классификаторам
    name = classifier.__class__.__name__ #Получение названия классификатора
    startTime = time.time() #Фиксирование времени начала тренировки, обучения алгоритма
    classifier.fit(X_train, y_train)  #Обучение алгоритма
    timeLearn = (time.time() - startTime) * 1000 #Фиксирование времени завершения работы алгоритма в миллисекундах

    startTime = time.time()#Фиксирование время начала предсказания алгоритма
    y_pred = classifier.predict(X_test) #Предсказывание
    timePredict = (time.time() - startTime) * 1000 #Фиксирование времени завершения работы алгоритма в миллисекундах

    accuracy = accuracy_score(y_test, y_pred) #Точность работы алгоритма
    print('Accuracy : %0.2f' % (accuracy))  #Вывод точности алгоритма

    print(classification_report(y_pred, y_test)) #Вывод отчета работы данного алгоритма


    entry = pd.DataFrame([[name, accuracy, timeLearn, timePredict]], columns=columns) #Создание строки с данными работы алгоритма, для вставки в таблицу результатов

    result = result.append(entry) #Вставка строки в таблицу
```

Вывод результатов работы.

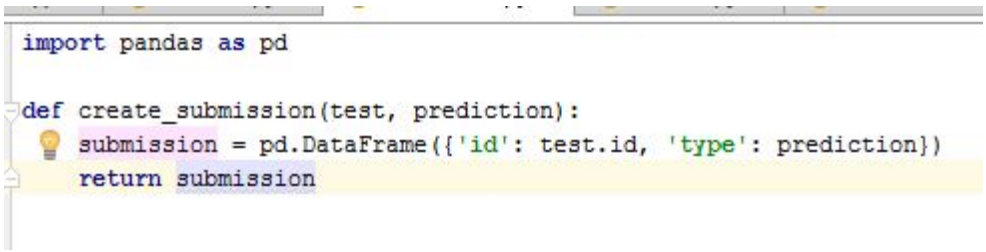
Classifier	Accuracy	Time Learn	Time Predict
RandomForestClassifier	0.666667	39.999962	0.000000
AdaBoostClassifier	0.573333	1292.000055	49.999952
DecisionTreeClassifier	0.600000	0.000000	0.000000
GradientBoostingClassifier	0.680000	1039.999962	9.999990
SVC	0.293333	19.999981	0.000000
KNeighborsClassifier	0.666667	0.000000	0.000000
LinearDiscriminantAnalysis	0.653333	0.000000	0.000000
QuadraticDiscriminantAnalysis	0.720000	0.000000	0.000000
MLPClassifier	0.720000	379.999876	0.000000

Для участия в конкурсе

Для участия в соревновании, необходимо, чтобы алгоритм выдавал в конечном итоге файл в виде submission.csv, в котором будут находиться предсказанные монстры.

Для этого был написан метод, который генерирует этот файл. Он принимает на вход тестовые данные, которые являются идентификаторами монстров и предсказанные названия этих монстров.

Библиотека pandas используется для создания двумерного массива, с названиями колонок id и type.



```
import pandas as pd

def create_submission(test, prediction):
    submission = pd.DataFrame({'id': test.id, 'type': prediction})
    return submission
```

Выводы

При выполнении данного задания, был получен такой результат. Первые три места по точности оказались:

QuadraticDiscriminantAnalysis
MLPClassifier
RandomForestClassifier

Эти три алгоритма наиболее точно выдавали результат. Но так как разница у них не особо большая, то при отправке полученного файла submission.csv получилось так, что самым лучшим оказался MLPClassifier. С его помощью был достигнут результат в конкурсе выше 200-го места. И его точность составила 0.745

Алгоритм RandomForestClassifier - хороший алгоритм для большого количества различных задач. Он работает на основе принципа “разделяй и властвуй” для ускорения вычислений и удобства. Также учитывает шумы в данных. Имеет большое количество параметров, с помощью которых можно сделать максимально точные вычисления, для каждой задачи.

Алгоритм MLPClassifier - весьма удобный и производительный. Он работает на основе нейронных сетей. У него есть такие функции, как ограничение по переобучению, то есть, если уже достигнут наиболее хороший вариант, то он не будет продолжать поиск, а остановит алгоритм. Это очень удобно, так как для большого числа итераций алгоритм может работать достаточно долго, но с помощью этой функции, можно получить результат намного быстрее.