Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу «Искусственный интеллект»
Тема: Линейные модели

Студент: К. А. Спиридонов
Преподаватель: Самир Ахмед
Группа: М8О-407Б-19
Дата:
Оценка:
Подпись:

Москва, 2022

Задача

Вы собрали данные и их проанализировали, визуализировали и представили отчет своим партнерам и спонсорам. Они согласились, что ваша задача имеет перспективу и продемонстрировали заинтересованность в вашем проекте. Самое время реализовать прототип! Вы считаете, что нейронные сети переоценены (просто боитесь признаться, что у вас не хватает ресурсов и данных), и считаете что за машинным обучением классическим будущее и потому собираетесь использовать классические модели. Вашим первым предположением является предположение, что данные и все в этом мире имеет линейную зависимость, ведь не зря же в конце каждой нейронной сети есть линейный слой классификации. В качестве первых моделей вы выбрали, линейную / логистическую регрессию и SVM. Так как вы очень осторожны и боитесь ошибиться, вы хотите реализовать случай, когда все таки мы не делаем никаких предположений о данных, и взяли за основу идею "близкие объекты дают близкий ответ"и идею, что теорема Байеса имеет ранг королевской теоремы. Так как вы не доверяете другим людям, вы хотите реализовать алгоритмы сами с нуля без использования scikit-learn (почти). Вы хотите узнать насколько хорошо ваши модели работают на выбранных вам данных и хотите замерить метрики качества. Ведь вам нужно еще отчитаться спонсорам!

Формально говоря вам предстоит сделать следующее: 1) реализовать следующие алгоритмы машинного обучения: Linear/ Logistic Regression, SVM, KNN, Naive Bayes в отдельных классах 2) Данные классы должны наследоваться от BaseEstimator и ClassifierMixin, иметь методы fit и predict 3) Вы должны организовать весь процесс предобработки, обучения и тестирования с помощью Pipeline (подробнее: https://scikit-learn.org/stable/modules/compose.html) 4) Вы должны настроить гиперпараметры моделей с помощью кросс валидации (GridSearchCV,RandomSearchCV, вывести и сохранить эти гиперпараметры в файл, вместе с обученными моделями 5) Проделать аналогично с коробочными решениями 6) Для каждой модели получить оценки метрик:Confusion Matrix, Accuracy, Recall, Precision, ROC_AUC curve 7) Проанализировать полученные результаты и сделать выводы о применимости моделей 8) Загрузить полученные гиперпараметры модели и обученные модели в формате pickle на гит вместе с jupyter notebook ваших экспериментов

Перед примененим алгоритмов к данным их надо немного обработать. Сделать то, что делал в лабе 0. И порешать проблему с перебалансировкой:

```
X_processed = full_processor.fit_transform(X)
y_processed = y.values.reshape(-1,1)
print('X Shape: ', X_processed.shape)
print('y shape: ', y_processed.shape)
```

```
X Shape:  (5109, 15)
y shape:  (5109, 1)
```

Воспользуемся RandomOverSampler для борьбы с несбалансированностью классов

1 Реализация алгоритмов и результаты

Logistic Regression

Реализация:

```
class My_LogisticRegression(BaseEstimator, ClassifierMixin):
    def __init__(self, lr=0.1, batch=10, epochs=1, alpha=0.0001):
        self.lr = lr
        self.batch = batch
        self.epochs = epochs
        self.alpha = alpha

    def fit(self, data, labels):
        self.w = np.random.normal(0, 1, (data.shape[1]+1,))
        data = np.concatenate((data, np.ones((data.shape[0],1))), axis=1)
        for _ in range(self.epochs):
            for i in range(self.batch, len(data), self.batch):
                data_batch = data[i-self.batch:i]
                labels_batch = labels[i-self.batch:i]

                pred = self.sigmoid(np.dot(self.w, data_batch.T))
                grad = 2 * self.alpha * self.w + np.dot(pred - labels_batch, data_batch)

                self.w -= self.lr * grad
        return self

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def predict(self, data):
        return (self.sigmoid(np.concatenate((data, np.ones((data.shape[0],1))), axis=1).dot(self.w)) > 0.5).astype('int64')
```

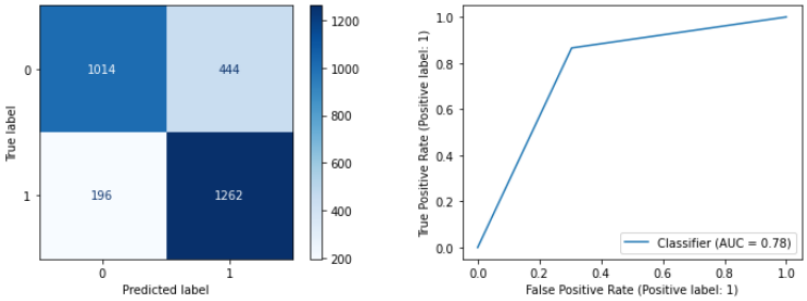
Результат применения:

```
grid_lr = GridSearchCV(Pipeline([('normalizer', Normalizer()), ('log', My_LogisticRegression())]),
                       {'log__lr': [0.1, 0.01, 0.001], 'log__epochs': [1, 10, 100],
                        'log__batch': [10, 100, 1000], 'log__alpha': [0.01, 0.001, 0.0001]})
grid_lr.fit(X_train, y_train)
print(grid_lr.best_params_, grid_lr.best_score_, sep='\n')
```

```
{'log__alpha': 0.0001, 'log__batch': 10, 'log__epochs': 100, 'log__lr': 0.01}
0.7771870812983532
```

```
[111] log = grid_lr.best_estimator_
      draw_scores(log, X_test, y_test)
```

```
Accuracy: 0.7805212620027435
Precision: 0.7397420867526378
Recall: 0.8655692729766804
```



KNN

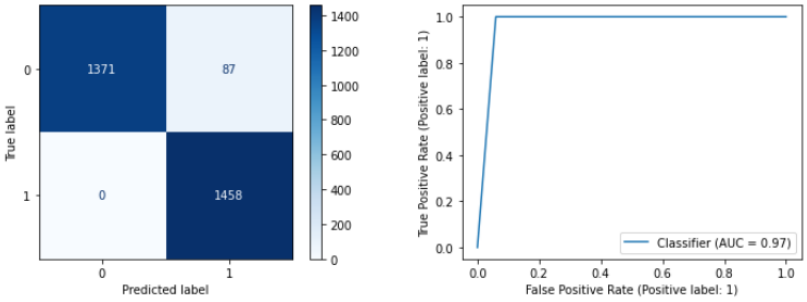
Реализация:

```
[47] class My_KNN(BaseEstimator, ClassifierMixin):
         def __init__(self, k=5):
             self.k = k

         def fit(self, data, labels):
             self.data = data
             self.labels = labels
             return self

         def predict(self, data):
             res = np.ndarray((data.shape[0],))
             for i, x in enumerate(data):
                 neighbors = np.argpartition(((self.data - data[i]) ** 2).sum(axis=1), self.k - 1)[:self.k]
                 values, counts = np.unique(self.labels[neighbors], return_counts=True)
                 res[i] = values[counts.argmax()]
             return res
```

Результат применения:

```
knn = grid_knn.best_estimator_
draw_scores(knn, X_test, y_test)
```

```
Accuracy: 0.970164609053498
Precision: 0.9436893203883495
Recall: 1.0
```



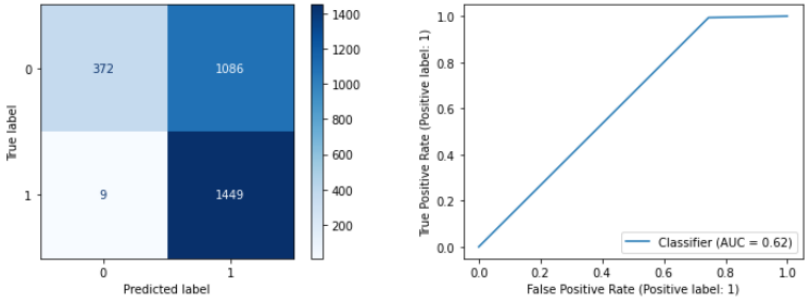
NaiveBayes

Реализация:

```
class My_GaussianNB(BaseEstimator, ClassifierMixin):
    def __init__(self):
        pass

    def fit(self, data, labels):
        self.data = data
        self.labels = labels
        self.means = []
        self.stds = []
        for c in np.unique(labels):
            self.means.append(data[labels == c,].mean(axis=0))
            self.stds.append(data[labels == c,].std(axis=0))
        self.classes = np.unique(labels, return_counts=True)[1] / len(labels)
        return self

    def predict(self, data):
        res = np.ndarray((data.shape[0],))
        for i, obj in enumerate(data):
            prob = np.array(self.classes)
            for j in range(len(self.classes)):
                prob[j] *= np.cumprod(1 / self.stds[j] / np.sqrt(2 * np.pi) * np.exp(((obj - self.means[j]) / self.stds[j]) ** 2 / -2))
            res[i] = prob.argmax()
        return res
```

Результат применения:

```
Accuracy: 0.6244855967078189
Precision: 0.5715976331360947
Recall: 0.9938271604938271
```



SVM

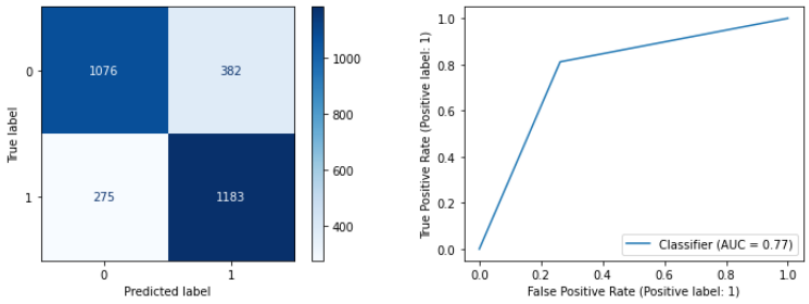
Реализация:

```python
class My_SVM(BaseEstimator, ClassifierMixin):
    def __init__(self, lr=0.1, batch=10, epochs=1, alpha=0.0001):
        self.lr = lr
        self.batch = batch
        self.epochs = epochs
        self.alpha = alpha

    def fit(self, data, labels):
        self.w = np.random.normal(0, 1, (data.shape[1]+1,))
        data = np.concatenate((data, np.ones((data.shape[0],1))), axis=1)
        labels = labels * 2 - 1
        for _ in range(self.epochs):
            for i in range(self.batch, len(data), self.batch):
                data_batch = data[i-self.batch:i]
                labels_batch = labels[i-self.batch:i]

                grad = 2 * self.alpha * self.w
                for i, x in enumerate(data_batch):
                    if 1 - x.dot(self.w) * labels_batch[i] > 0:
                        grad -= x * labels_batch[i]

                self.w -= self.lr * grad
        return self

    def predict(self, data):
        return (np.sign(np.concatenate((data, np.ones((data.shape[0],1))), axis=1).dot(self.w)) + 1) / 2
```

Результат применения:

```
svm = grid_svm.best_estimator_
draw_scores(svm, X_test, y_test)
```

```
Accuracy: 0.7746913580246914
Precision: 0.7559105431309904
Recall: 0.8113854595336076
```



2 Выводы

Выполнив лабораторную работу, я познакомился с важными алгоритмами машинного обучения: kNN, SVM, LogisticRegression, Naive Bayes. После их реализации я смог лучше понять их принцип. Что понравилось в этой лабораторной, так это то, что задача приближена к реальным задачам и данную модель(возможно) можно применять на практике. Так же порадовал результат алгоритма kNN с помощью него удалось достичь точности 97