Московский Авиационный Институт
(Национальный исследовательский университет)

Кафедра 806

Отчет
по лабораторной работе №1
по курсу «Искусственный интеллект»

Студент: Ильиных В. М.

Группа: М8О-301Б-19

Москва, 2022

Задача

1. Реализовать алгоритмы Linear/Logistic Regression, KNN, SVM, Naive Bayes в отдельных классах
2. Классы должны наследоваться от BaseEstimator и ClassifierMixin
3. Организовать процесс предобработки, обучения и тестирования с помощью Pipeline
4. Настроить гиперпараметры с помощью кросс-валидации (GridSearchCV, RandomSearchCV), вывести и сохранить эти гиперпараметры в файл вместе с обученными моделями
5. Для каждой модели получить оценки метрик: Confusion Matrix, Accuracy, Recall, Precision, ROC_AUC curve
6. Проанализировать полученные результаты и сделать выводы о применимости моделей

Ход работы

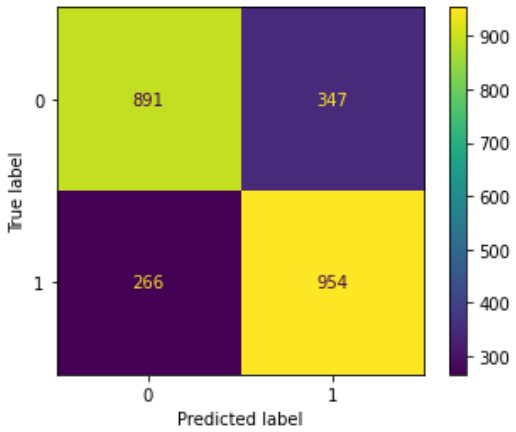
Логистическая регрессия имеет 4 параметра, значит используем поиск по сетке, но сначала нормализуем.

```
class Logistic_Regression(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):
            warnings.filterwarnings('ignore')
            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')
```

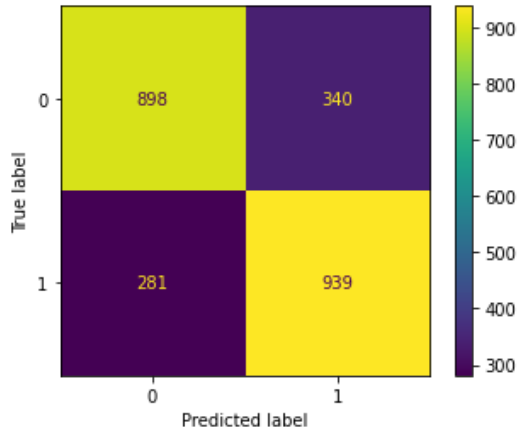
Результаты:

```
Accuracy: 0.7506102522375916
Precision: 0.7332820906994619
Recall: 0.7819672131147541
```



Результаты логистической регрессии из sklearn

```
Accuracy: 0.7473555736371034
Precision: 0.7341673182173573
Recall: 0.769672131147541
```

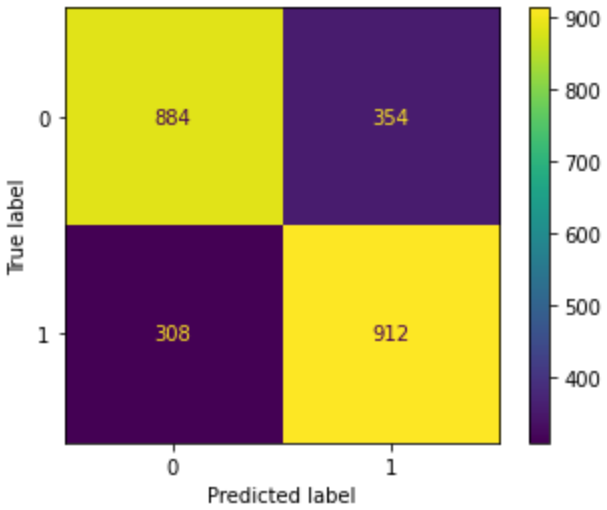


Метод ближайших соседей – KNN – имеет единственный параметр – собственно количество соседей. Для вычисления расстояние используется евклидова метрика.

```
class 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
```

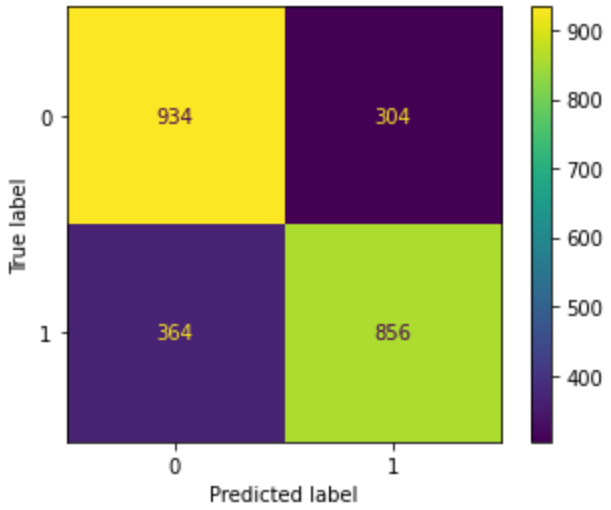
Результаты:

```
Accuracy: 0.7306753458096013
Precision: 0.7203791469194313
Recall: 0.7475409836065574
```



Результаты KneighborsClassifier из sklearn

```
Accuracy: 0.7282343368592351
Precision: 0.7379310344827587
Recall: 0.7016393442622951
```



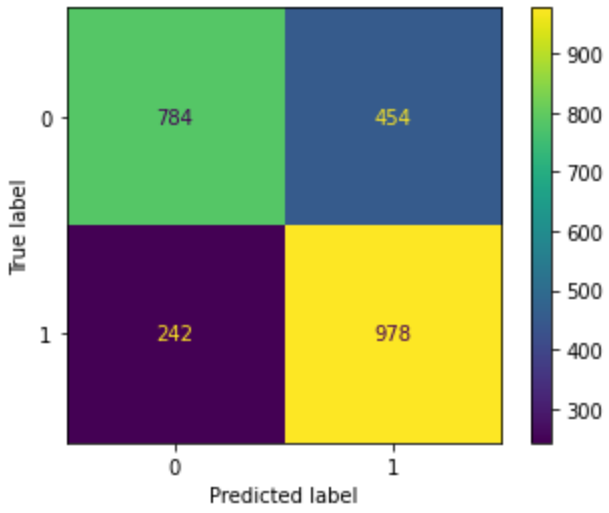
Метод Naive Bayes:

```
class NaiveBayes(BaseEstimator, ClassifierMixin):
    def __init__(self, bins):
        self.bins = bins
        pass

    def fit(self, data, labels):
        self.data = data
        self.labels = labels
        self.classes = []
              for j in np.unique(labels):

                  self.classes.append([])
                  for i in range (data.shape[1]):
                      self.classes[j].append([*np.histogram(data[labels == j, i], bins = self.bins)])
                      self.classes[j][-1][0] = self.classes[j][-1][0].astype('float64') / len(data[labels == j, i])

            self.prclasses = np.unique(labels, return_counts = True)[1] / len(labels)

        def predict(self, maindata):
```

```
            res = np.ndarray((maindata.shape[0],))
            for j, data in enumerate(maindata):
                maximum = 0
                ans = 0
                for i in range(len(self.classes)):
                    p = self.prclasses[i]
                    for k in range(len(self.classes[i])):
                        ind = np.digitize(data[k],
self.classes[i][k][1])

                        if ind >= len(self.classes[i][k][1]) or ind <=
0:
                            p = 0
                        else:
                            p *= self.classes[i][k][0][ind - 1]

                    if p > maximum:
                        maximum = p
                        ans = i
                res[j] = ans
            return res
```

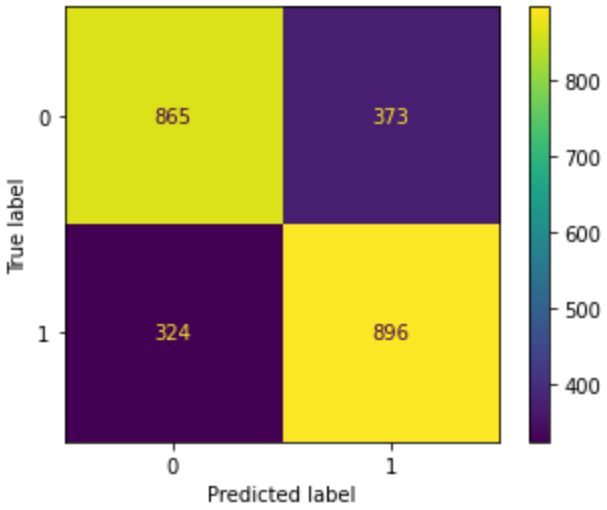
Результаты:

```
Accuracy: 0.7168429617575265
Precision: 0.6829608938547486
Recall: 0.8016393442622951
```



Результаты для реализации из sklearn:

```
Accuracy: 0.7164361269324654
Precision: 0.7060677698975572
Recall: 0.7344262295081967
```



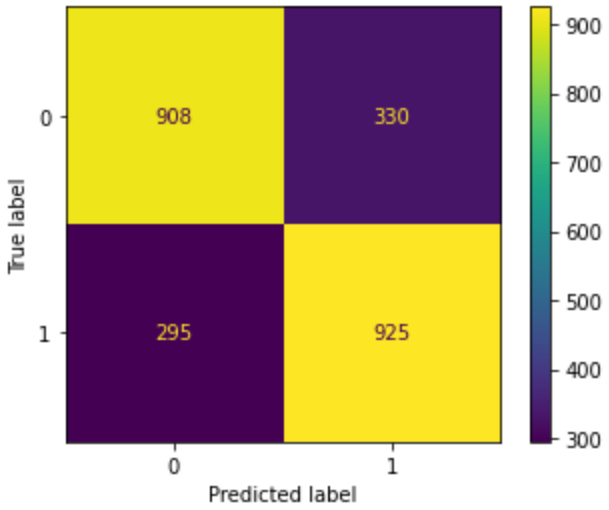
Последний метод – SVM имеет 4 параметра, используем поиск по сетке:

```python
class 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]
```

```
21.
22.                     self.w -= self.lr * grad
23.             return self
24.
25.         def predict(self, data):
26.             return (np.sign(np.concatenate((data,
    np.ones((data.shape[0],1))), axis=1).dot(self.w)) + 1) / 2
```

Результаты:

```
Accuracy: 0.7457282343368592
Precision: 0.7370517928286853
Recall: 0.7581967213114754
```



Выводы

Как видно из результатов, итоги не самые плохие – от 70 до 80%, однако и не самые хорошие. Лучше всего себя показал метод SVM, однако метрика Recall самая высокая – в районе 80% - у метода Naive Bayes. Реализации из sklearn оказались местами хуже, нежели написанные с нуля.

Можно сделать вывод, что данная задача не очень хорошо решается машинным обучением.