Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**
**«Машинное обучение»**

# Линейные модели

Студент: Шавандрин Фёдор Михайлович
Группа: М80-308Б-19
Дата: 29.05.2022
Оценка: _____
Подпись: _____

Москва, 2022

# 1. Постановка задачи

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
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

# 2. Подготовка данных

Для начала необходимо подготовить данные для обучения:

- Категориальные фичи преобразуем с помощью *ohe (OneHotEncoder)*. Если признак бинарный, то будем оставлять только один столбец для него.
- Количественные признаки преобразуем с помощью MinMaxScaler.

```
data_preprocessing = ColumnTransformer([
    ('ohe', OneHotEncoder(drop='if_binary'), categorical_features),
    ('minmax', MinMaxScaler(), [feature for feature in numerical_features])
])
```

Датасет имеет 4 класса ценовой категории телефонов (0,1,2,3). Изменим датасет так, чтобы новая ценовая категория 0 будет соответствовать старым классам 0 и 1, а категория 1 - соответственно 2 и 3.
Разделим данные на трейн и тест с помощью функции train_test_split. На тестовую часть оставим 30% данных.

# 3. Подсчет метрик

Сразу определим функцию для оценивания качества моделей. Будем считать метрики accuracy, precision, recall, rocauc и будем строить confusion matrix.

```
def get_metrics(model, X, y_true, threshold=0.5, use_probas=True):

    if use_probas:
        y_pred_probas = model.predict_proba(X)
        if len(y_pred_probas.shape) == 2:
            y_pred_probas = y_pred_probas[:, 1]
        y_pred = y_pred_probas > threshold
    else:
        y_pred = model.predict(X)

    print('Accuracy = ', accuracy_score(y_true, y_pred))
```

```
    print('Precision = ', precision_score(y_true, y_pred))
    print('Recall = ', recall_score(y_true, y_pred))
    if use_probas:
        print('ROC AUC = ', roc_auc_score(y_true, y_pred_probas))
    print('Confusion matrix:')
    print(confusion_matrix(y_true, y_pred))
```

## 4. Обучение и валидация моделей

- **Логистическая регрессия**

```
class My_log_regression(BaseEstimator, ClassifierMixin):
    def __init__(self, epochs=10, lr=0.1, batch_size=256):
        self.w = None
        self.epochs = epochs
        self.lr = lr
        self.batch_size = batch_size

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        n, k = X.shape

        if self.w is None:
            np.random.seed(0xDEAD)
            self.w = np.random.randn(k + 1)

        X = np.concatenate((np.ones((n, 1)), X), axis=1)
        for i in range(self.epochs):
            for j in range(0, len(X), self.batch_size):
                X_batch = X[j:j+self.batch_size]
                y_batch = y[j:j+self.batch_size]

                y_pred = self._predict_proba_internal(X_batch)
                    self.w -= self.lr * self._get_gradient(X_batch, y_batch,
y_pred)

        return self

    def _get_gradient(self, X_batch, y_batch, y_pred):
        gradient = X_batch.T @ (y_pred - y_batch)
        return gradient

    def predict_proba(self, X):
        X = check_array(X)

        n = X.shape[0]
        X = np.concatenate((np.ones((n, 1)), X), axis=1)
        return self._sigmoid(np.dot(X, self.w))

    def _predict_proba_internal(self, X):
        return self._sigmoid(np.dot(X, self.w))

    def predict(self, X, threshold=0.5):
        return self.predict_proba(X) > threshold

    def _sigmoid(self, a):
        return 1. / (1 + np.exp(-a))
```

Получили следующие результаты:

```
Accuracy =  0.9433333333333334
Precision =  0.9039039039039038
Recall =  0.9933993399339934
ROC AUC =  0.9973775155293307
Confusion matrix:
[[265  32]
 [  2 301]]
```

Реализация лог. регрессии из sklearn дала следующие результаты с добавим весов к классам (class_weight='balanced'):

```
Accuracy =  0.9833333333333333
Precision =  0.9834983498349835
Recall =  0.9834983498349835
ROC AUC =  0.9927659432609928
Confusion matrix:
[[292   5]
 [  5 298]]
```

Как видно по результатам, моя логистическая регрессия работает почти так же, как и библиотечная.

- **SVM**

линейный SVM с использованием soft margin loss

```
class My_svm(ClassifierMixin, BaseEstimator):
    def __init__(self, epochs=10, lr=0.1, alpha=0.1):
        self.w = None
        self.epochs = epochs
        self.lr = lr
        self.alpha = alpha

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        y = np.where(y == 1, 1, -1)
        n, k = X.shape

        if self.w is None:
            np.random.seed(66)
            self.w = np.random.randn(k + 1)

        X = np.concatenate((np.ones((n, 1)), X), axis=1)
        for i in range(self.epochs):
            for j, x in enumerate(X):
                margin = y[j] * np.dot(self.w, x)
                if margin >= 1:
                    self.w -= self.lr * self.alpha * self.w / self.epochs
                else:
                    self.w += self.lr * (y[j] * x - self.alpha * self.w /
self.epochs)
        return self

    def predict(self, X):
        X = check_array(X)
        n, k = X.shape
```

```
        X = np.concatenate((np.ones((n, 1)), X), axis=1)
        y = np.ndarray((n))

        for i, elem in enumerate(X):
            prediction = np.dot(self.w, elem)
            if prediction > 0:
                y[i] = 1
            else:
                y[i] = 0
        return y


    def _hinge_loss(self, x, y):
        return max(0, 1 - y * np.dot(x, self.w))


    def _soft_margin_loss(self, x, y):
        return self._hinge_loss(x, y) + self.alpha * np.dot(self.w, self.w)
```

Получили следующие результаты:

```
Accuracy =  0.9533333333333334
Precision =  1.0
Recall =  0.9075907590759076
Confusion matrix:
[[297   0]
 [ 28 275]]
```

Результаты модели SVM из sklearn:

```
Accuracy =  0.9833333333333333
Precision =  0.9834983498349835
Recall =  0.9834983498349835
Confusion matrix:
[[292   5]
 [  5 298]]
```

Результаты моей SVM почти совпали SVM из sklearn.

- **KNN**

```
from sklearn.metrics import euclidean_distances

class My_knn(ClassifierMixin, BaseEstimator):
    def __init__(self, k = 1):
        self.k = k

    def fit(self, X, y):
        # Check that X and y have correct shape
        X, y = check_X_y(X, y)
        # Store the classes seen during fit
        self.classes_ = unique_labels(y)

        self.X_ = X
        self.y_ = y
        # Return the classifier
        return self

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'y_'])
```

```
            # Input validation
            X = check_array(X)

            y = np.ndarray((X.shape[0],))
            for (i, elem) in enumerate(X):
                distances = euclidean_distances([elem], self.X_)[0]
                neighbors = np.argpartition(distances, kth = self. k - 1)
                k_neighbors = neighbors[:self.k]
                labels, cnts = np.unique(self.y_[k_neighbors], return_counts =
   True)
                y[i] = labels[cnts.argmax()]
            return y
```

Получили следующие результаты:

```
Accuracy =  0.7316666666666667
Precision =  0.7448275862068966
Recall =  0.7128712871287128
Confusion matrix:
[[223  74]
 [ 87 216]]
```

Библиотечная версия алгоритма:

```
Accuracy =  0.7316666666666667
Precision =  0.7448275862068966
Recall =  0.7128712871287128
ROC AUC =  0.8100532275449768
Confusion matrix:
[[223  74]
 [ 87 216]]
```

- **Наивный алгоритм Байеса**

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

    def fit(self, X, y):
        X, y = check_X_y(X, y)
        labels, counts = np.unique(y, return_counts=True)
        self.labels = labels
        self.freq = np.array([cnt / y.shape[0] for cnt in counts])
        self.means = np.array([X[y == label].mean(axis = 0) for label in
labels])
        self.stds = np.array([X[y == label].std(axis = 0) for label in
labels])

        return self

    def predict_proba(self, X):
        X = check_array(X)
        y = np.zeros(X.shape[0])
        for i, x in enumerate(X):
            cur_freq = np.array(self.freq)
            for j in range(len(self.labels)):
                p = np.array([self._gaussian(self.means[j][k], self.stds[j]
[k], x[k]) for k in range(X.shape[1])])
                cur_freq[j] *= np.prod(p)
```

```
            y[i] = cur_freq[1]
        return y

    def predict(self, X, threshold=0.5):
        return self.predict_proba(X) > threshold

    def _gaussian(self, mu, sigma, x0):
        return np.exp(-(x0 - mu) ** 2 / (2 * sigma)) / np.sqrt(2.0 * np.pi *
sigma)
```

Проверим результаты:

```
Accuracy =  0.505
Precision =  0.505
Recall =  1.0
ROC AUC =  0.7696769676967696
Confusion matrix:
[[  0 297]
 [  0 303]]
```

Попробуем использовать GaussianNB из sklearn.

```
Accuracy =  0.8383333333333334
Precision =  1.0
Recall =  0.6798679867986799
ROC AUC =  0.9816981698169817
Confusion matrix:
[[297   0]
 [ 97 206]]
```

Проанализировав результаты работы, видно, что моя версия наивного алгоритма Байеса отработала хуже, чем библиотечная версия. Это может быть связано с тем, что я использовал произведение вероятностей в отличие от версии sklearn, где используется минус логарифм их суммы.

## 5. Вывод

В данной лабораторной работе я реализовал некоторые линейные алгоритмы машинного обучения - логистическую регрессию, SVM, KNN и наивного Байеса. Я попробовал обучить каждую из этих моделей на своем датасете и посчитал метрики для каждой. Затем сравнил метрики моих моделей и моделей из sklearn и сделал вывод о том, результаты метрик почти совпадают за исключением наивного алгоритма Байеса. Причины, почему так произошло, описал выше в результатах работы наивного алгоритма Байеса.