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**
2. Реализовать следующие алгоритмы машинного обучения: Linear/ Logistic Regression, SVM, KNN, Naive Bayes в отдельных классах
3. Данные классы должны наследоваться от BaseEstimator и  ClassifierMixin, иметь методы fit и predict
4. Вы должны организовать весь процесс предобработки, обучения и тестирования с помощью Pipeline
5. Вы должны настроить гиперпараметры моделей с помощью кросс валидации (GridSearchCV, RandomSearchCV) вывести и сохранить эти гиперпараметры в файл, вместе с обученными моделями
6. Проделать аналогично с коробочными решениями
7. Для каждой модели получить оценки метрик: Confusion Matrix, Accuracy, Recall, Precision, ROC\_AUC curve
8. Проанализировать полученные результаты и сделать выводы о применимости моделей
9. Загрузить полученные гиперпараметры модели и обученные модели в формате pickle на гит вместе с jupyter notebook ваших экспериментов
10. **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.

1. **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))

1. **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))

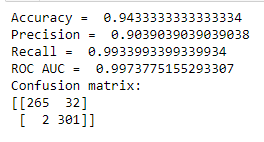
def predict(self, X, threshold=0.5):

return self.predict\_proba(X) > threshold

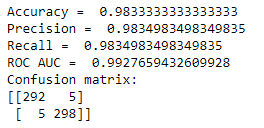
def \_sigmoid(self, a):

return 1. / (1 + np.exp(-a))

Получили следующие результаты:



Реализация лог. регрессии из sklearn дала следующие результаты с добавим весов к классам (class\_weight=’balanced’):



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

* **SVM**

линейный SVM c 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

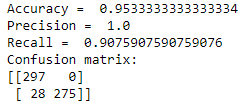
def \_hinge\_loss(self, x, y):

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

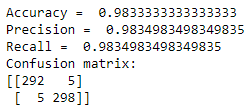
def \_soft\_margin\_loss(self, x, y):

return self.\_hinge\_loss(x, y) + self.alpha \* np.dot(self.w, self.w)

Получили следующие результаты:



Результаты модели SVM из sklearn:



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)

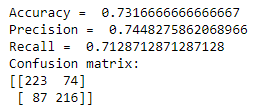
k\_neighbors = neighbors[:self.k]

labels, cnts = np.unique(self.y\_[k\_neighbors], return\_counts = True)

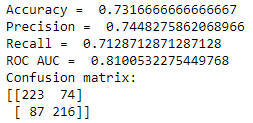
y[i] = labels[cnts.argmax()]

return y

Получили следующие результаты:



Библиотечная версия алгоритма:



* **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

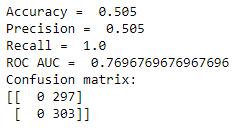
def predict(self, X, threshold=0.5):

return self.predict\_proba(X) > threshold

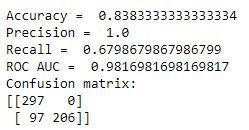
def \_gaussian(self, mu, sigma, x0):

return np.exp(-(x0 - mu) \*\* 2 / (2 \* sigma)) / np.sqrt(2.0 \* np.pi \* sigma)

Проверим результаты:



Попробуем использовать GaussianNB из sklearn.



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.

1. **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.