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: Моисеенков Илья Павлович

Группа: М80 – 308Б-19

Дата: 24.05.2022

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 признаки преобразуем с помощью one-hot encoding. Если фича бинарная, то будем оставлять только одну фичу для нее.
* BMI (индекс массы тела) приведем к стандартному нормальному распределению (т.к. этот признак изначально был распределен нормально) с помощью Standard Scaler.
* остальные количественные признаки преобразуем с помощью MinMaxScaler.

data\_preprocessing = ColumnTransformer([

('ohe', OneHotEncoder(drop='if\_binary'), categorical\_features),

('stdscale', StandardScaler(), ['BMI']),

('minmax', MinMaxScaler(), [feat for feat in numerical\_features if feat != 'BMI'])

])

Разделим данные на трейн и тест с помощью функции train\_test\_split. На тестовую часть оставим 20% данных.

1. **Подсчет метрик**

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

Важный момент: мы решаем задачу предсказания вероятности сердечного приступа. Нам важно максимизировать реколл. Поэтому будем дополнительно оценивать соотношение пресижена и реколла с помощью PR-curve. Зафиксируем реколл=0.7 и будем оценивать модель при таком реколле.

Для SVM нет возможности получить вероятность принадлежности к классу, поэтому учтем это в функции.

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))

if use\_probas:

precision, recall, thresholds = precision\_recall\_curve(y\_true, y\_pred\_probas)

plt.figure(figsize=(15, 8))

plt.xlabel('Recall')

plt.ylabel('Presicion')

plt.title('Precision-recall curve')

plt.xticks(np.arange(0, 1.1, 0.1))

plt.yticks(np.arange(0, 1.1, 0.1))

plt.grid()

plt.plot(recall, precision)

1. **Обучение и валидация моделей**

* **KNN**

from sklearn.metrics import euclidean\_distances

class MyKNN(BaseEstimator, ClassifierMixin):

def \_\_init\_\_(self, n\_neighbors=5):

self.n\_neighbors = n\_neighbors

def fit(self, X, y):

# check equal shapes

X, y = check\_X\_y(X, y)

# remember x and y

self.X\_ = X

self.y\_ = y

return self

def predict\_proba(self, X):

"""

Get P(y == 1 | X)

"""

# check that X is a correct array

X = check\_array(X)

y = np.ndarray((X.shape[0]))

for i, elem in enumerate(X):

# get distances and labels

distances = euclidean\_distances([elem], self.X\_)[0]

distances\_with\_labels = np.stack((distances, self.y\_), axis=1)

distances\_with\_labels.sort(axis=0)

# get k nearest neighbors and count their labels

k\_neighbors = distances\_with\_labels[:self.n\_neighbors]

labels, counts = np.unique(k\_neighbors[:, 1], return\_counts=True)

for j, label in enumerate(labels):

if label == 1:

proba1 = counts[j] / self.n\_neighbors

break

else:

# no neighbors with label = 1

proba1 = 0

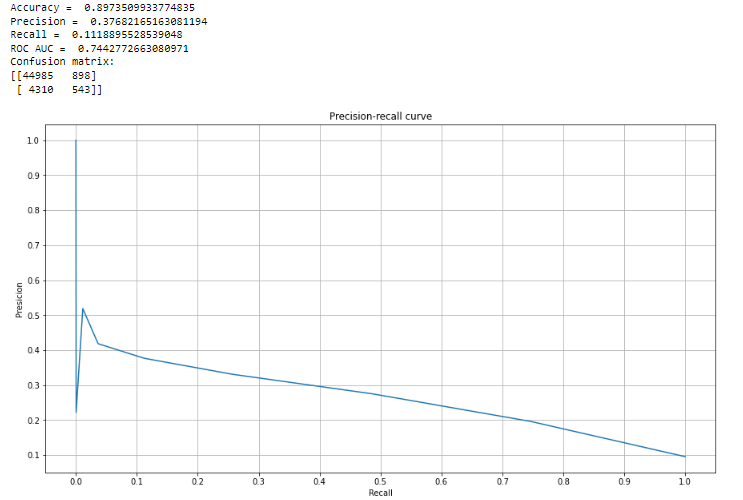
y[i] = proba1

return y

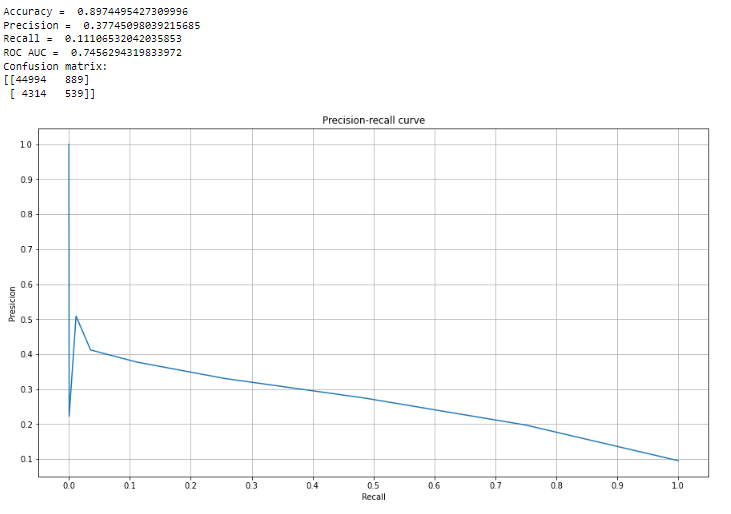
def predict(self, X, threshold=0.5):

return self.predict\_proba(X) > threshold

Получили следующие результаты:



Для реализации из склерна получили следующие результаты:



*KNN очень долго работает, поэтому я не стал подбирать порог, чтобы получить реколл=0.7.*

При реколле 0.7 имеем пресижен, примерно равный 0.21. Результаты моей модели и модели склерна оказались почти одинаковые.

* **Logistic Regression**

Буду реализовывать логистическую регрессию через стохастический градиентный спуск.

class MyLogisticRegression(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)

# weights

self.w = np.random.randn(k + 1)

X = np.concatenate((np.ones((n, 1)), X), axis=1) # add bias as feature

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):

"""

Get gradient for logistic regression

"""

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):

"""

This function is similar to predict\_proba, but we don't concatenate bias here.

It is used for fitting.

"""

return self.\_sigmoid(np.dot(X, self.w))

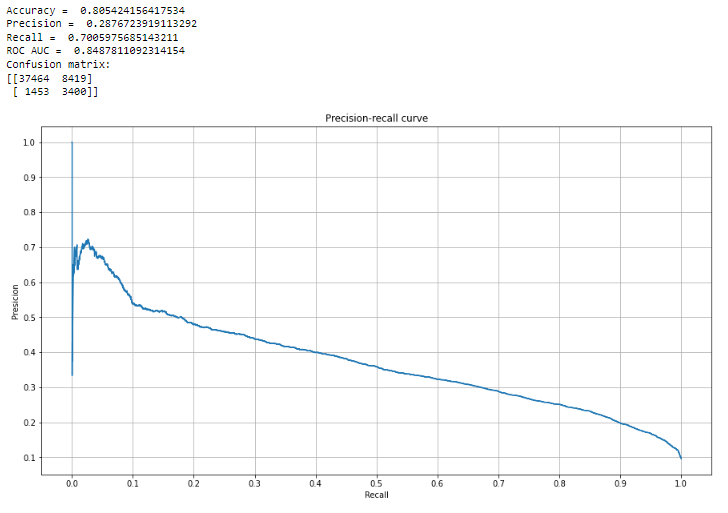
def predict(self, X, threshold=0.5):

return self.predict\_proba(X) > threshold

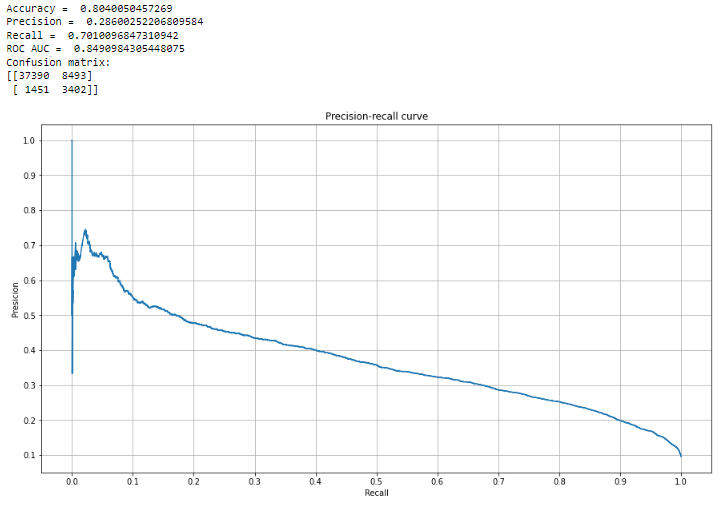
def \_sigmoid(self, a):

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

Получили следующие результаты:



Попробуем использовать реализацию из склерна. Добавим веса к классам (class\_weight=’balanced’). Получили следующие результаты:



Результаты моей регрессии совпали (почти) с библиотечной.

* **Naive Bayes**

Предположим, что все фичи имеют нормальное распределение.

class MyNaiveBayes(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

# remember frequencies, means and standart deviations. we'll need it in bayes theorem

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(label[j]|X)

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

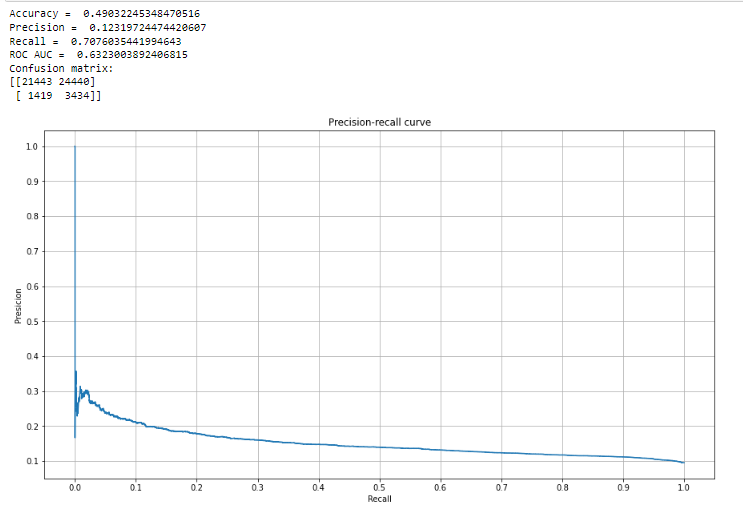
def \_gaussian(self, mu, sigma, x0):

# X ~ N(mu, sigma)

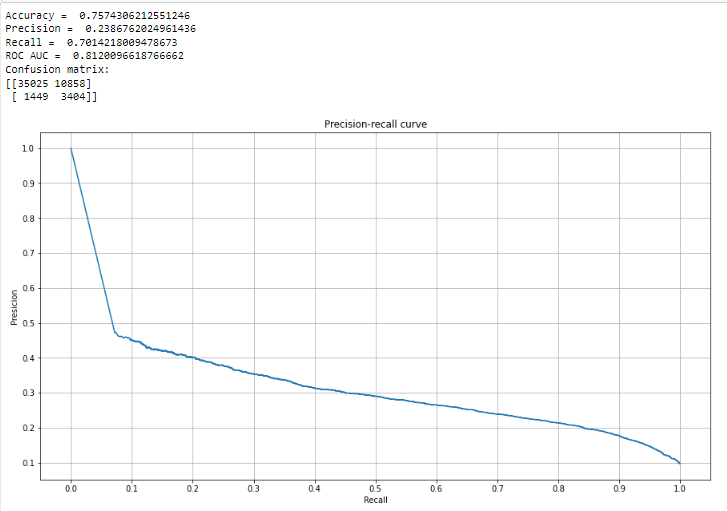
# counts F(x0), where F is distribution function

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

Получили следующие результаты:



Попробуем библиотечную версию алгоритма.



Видим, что библиотечный алгоритм справился с задачей лучше моего. Это возможно по нескольким причинам. Во-первых, в склерновской версии я использовал веса для классов, чтобы побороть несбалансированность. Во-вторых, вероятно, в библиотечной версии вместо произведения вероятностей используется минус логарифм их суммы. В моей версии я решил пойти традиционным путем и использовать произведение (хотя это не всегда хорошо и может привести к проблемам из-за произведения бесконечно малых чисел с плавающей точкой).

* **SVM**

Попробуем реализовать линейный SVM. Будем использовать soft margin loss.

class MySVM(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 # regularization parameter

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(0xDEAD)

# weights

self.w = np.random.randn(k + 1)

X = np.concatenate((np.ones((n, 1)), X), axis=1) # add bias as feature

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):

# no predict proba here

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

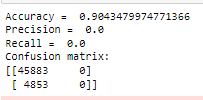
def \_hinge\_loss(self, x, y):

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

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

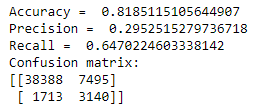
return self.\_hinge\_loss(x, y) + self.alpha \* np.dot(self.w, self.w)

Проверим результаты:



К сожалению, метод опорных векторов не смог построить хорошую разделяющую линейную гиперплоскость. Поэтому ни одна строка данных не получила класс 1 от модели. Я проверил работу своего алгоритма на более простом датасете - там все работало. Значит для СВМа моя несбалансированная задача оказалась слишком сложной.

Попробуем использовать LinearSVC из sklearn.



Балансировка классов снова помогла, мы получили какой-то более-менее приемлемый результат.

1. **Вывод**

В данной 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.

К сожалению, из-за несбалансированности в данных и из-за линейной неразделимости я не смог получить хороших результатов. Хотя я и получал всегда высокий эккюраси, соотношение пресижена и реколла оставляло желать лучшего.

Наилучший результат я смог получить при использовании логистической регрессии. Там при реколле 0.7 я смог получить пресижен 0.28. Результат пока не очень хороший. Нужно использовать более сложные модели, чтобы улучшить его. Поэтому я хочу попробовать деревянные модели. Но это уже в следующей лабе..