Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Машинное обучение»**

Студент: Кузьмичев Александр Николаевич

Группа: М80 – 306Б-18

Преподаватель: Ахмед Самир Халид

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Задание**

Найти себе набор данных (датасет), для следующей лабораторной работы, и

проанализировать его. Выявить проблемы набора данных, устранить их.

Визуализировать зависимости, показать распределения некоторых признаков.

Реализовать алгоритмы К ближайших соседа с использованием весов и наивный

Байесовский классификатор и сравнить с реализацией библиотеки sklearn.

**Описание датасета**

Для данной лабораторной работы я использовал датасет в формате csv, содержащий информацию о 5172 электронных письмах. Отличительной особенностью датасета является большое число столбцов в таблице (3002). Датасет взят с сайта kaggle.com, он доступен по ссылке[Email Spam Classification Dataset CSV | Kaggle](https://www.kaggle.com/balaka18/email-spam-classification-dataset-csv).

**Процесс очистки датасета**

Очистка датасета, как таковая, не потребовалась. Пропусков в данных не было, что видно из тепловой карты.

**Алгоритм KNN (K-nearest neighbors)**

**kNN** расшифровывается как k-Nearest Neighbors — это один из самых простых алгоритмов классификации, также иногда используемый в задачах регрессии.

Задача классификации в машинном обучении — это задача отнесения объекта к одному из заранее определенных классов на основании его формализованных признаков. Каждый из объектов в этой задаче представляется в виде вектора в N-мерном пространстве, каждое измерение в котором представляет собой описание одного из признаков объекта.

Для обучения классификатора необходимо иметь набор объектов, для которых заранее определены классы. Это множество называется обучающей выборкой, её разметка производится вручную, с привлечением специалистов в исследуемой области

Для классификации каждого из объектов тестовой выборки необходимо последовательно выполнить следующие операции:

* Вычислить расстояние до каждого из объектов обучающей выборки
* Отобрать k объектов обучающей выборки, расстояние до которых минимально
* Определить класс классифицируемого объекта как класс, наиболее часто встречающийся среди k ближайших соседей

Листинг кода с моей реализацией алгоритма KNN:

*class My\_knn\_with\_weights:*

*def \_\_init\_\_(self, neighbours):*

*self.K = neighbours*

*def fit(self, X\_train, y\_train):*

*self.X\_train = np.array(X\_train).copy()*

*self.y\_train = np.array(y\_train).copy()*

*def predict(self, X\_test):*

*X\_test = np.array(X\_test).copy()*

*y\_predicted = []*

*for row\_to\_predict in X\_test:*

*# расстояния до точек из тренировочной выборки в паре с классом*

*dists = []*

*for i in range(len(self.X\_train)):*

*dist = 0*

*for j in range(len(self.X\_train[i])):*

*dist += pow(row\_to\_predict[j] - self.X\_train[i, j], 2)*

*dists.append([math.sqrt(dist), self.y\_train[i]])*

*# К ближайших соседей*

*knn = sorted(dists, key=lambda x: x[0])*

*knn = knn[0 : self.K]*

*div\_dist = list()*

*for dist in knn:*

*div\_dist.append(1/dist[0])*

*W = list()*

*i = 0*

*for n in div\_dist:*

*W.append([n/sum(div\_dist), knn[i][1]])*

*i += 1*

*unique\_classes = np.unique(self.y\_train)*

*total\_probability = {}*

*for classs in unique\_classes:*

*total\_probability[classs] = 0*

*for weights in W:*

*total\_probability[weights[1]] += weights[0]*

*# максимальная вероятность*

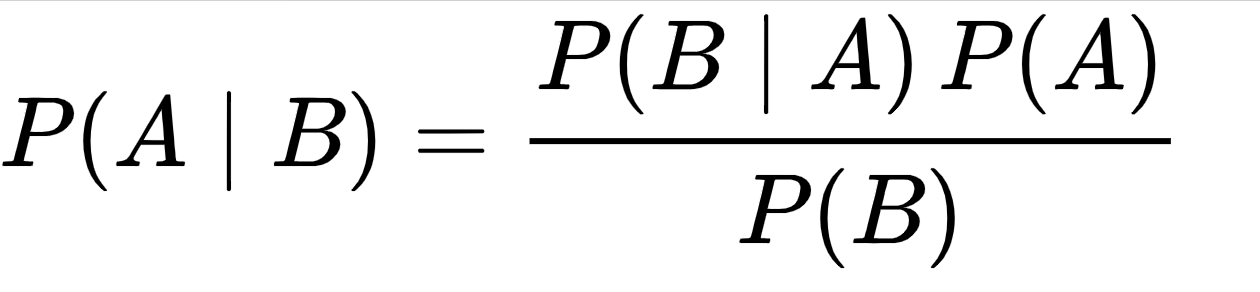
*y\_predicted.append(max(total\_probability, key=total\_probability.get))*

*y\_predicted = np.array(y\_predicted)*

*return y\_predicted*

**Классификатор Байеса**

Классификатор Байеса - простой вероятностный [классификатор](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8), основанный на применении теоремы Байеса.



Ниже – листинг кода с моей реализацией наивного классификатора Байеса.

*class My\_naive\_Bayes\_classifier:*

*def probability(self, i, x):*

*# вероятность в предположении, что фичи имеют нормальное распределение*

*return (1 / np.sqrt(2 \* np.pi \* self.variances[i])) \* \*

*np.exp((-1/2) \* (pow(x-self.expectations[i], 2)) / (2 \* self.variances[i]))*

*# результирующая вероятность*

*def result(self, x):*

*P = []*

*for i in range(self.num\_of\_classes):*

*P.append(np.log(self.p[i]) + np.sum(np.log(self.probability(i, x))))*

*predicted\_class = self.classes[np.argmax(P)]*

*return predicted\_class*

*def fit(self, X\_train, y\_train):*

*self.classes = np.unique(y\_train)*

*self.num\_of\_classes = len(self.classes)*

*# матожидания и дисперсии для столбцов таблицы (группируем по классам)*

*self.expectations = np.array(X\_train.groupby(y\_train).apply(np.mean))*

*self.variances = np.array(X\_train.groupby(y\_train).apply(np.var))*

*# априорная вероятность того, что рандомная строка принадлежит классу*

*self.p = np.array(X\_train.groupby(y\_train).apply(lambda col: len(col)) / len(y\_train))*

*def predict(self, X\_test):*

*y\_predicted = []*

*for x in np.array(X\_test):*

*y\_predicted.append(self.result(x))*

*y\_predicted = np.array(y\_predicted)*

*return y\_predicted*

**Выводы**

Выполнив первую лабораторную работу по машинному обучению, я познакомился с Jupyter Notebook, а также с двумя базовыми алгоритмами классификации. Так получилось, что до этого я имел весьма общее представления о ML. Данная лабораторная работа помогла разобраться мне в его основах. Думаю, что я продолжу изучать машинное обучение самостоятельно.