4 часть расчетного, Корнеич Никита 20931

Результаты:

```
4.1. Примеры предсказаний (0-setosa, 1-versicolor):

Для 1.0 [1]
Для 1.9 [1]
Для 5.1 [0]
Для 6.0 [0]
4.2. Оценить точность классификации при случайном разбиении выборки на обучающую (80%) и контрольную выборки (20%)
Точность: 0.856
Точность предсказания высокая поскольку входные данные сильно отличаются
друг от другаи их легко классифицировать.
```

```
def main():
    # 8 Вариант Корнеич Никита 20931
    # Iris Setosa, Iris virginica, Ширина лепестка
    df = pd.read_csv("iris.data", delimiter=',')
    df = df[df['class'].isin(['Iris-setosa', 'Iris-versicolor'])]

    # Выбираем только столбец с шириной лепестка

    # Это данные на основе которых будет обучаться модель
    X = df["width_of_petal"]

    xx = []
    for i in range(len(X)):
        xx.append([X[i]])


    # Преобразовываем лист в np.array
    X = np.array(xx)

    zeros = np.zeros(50, dtype='int')
    ones = np.ones(50, dtype='int')

    # Это данные на основе которых будет проверяться предсказания

    targets = np.concatenate((zeros, ones))

    # Делим данные на 80% тренировочных и 20% тестовых
    x_train, x_test, y_train, y_test = model_selection.train_test_split(X,
targets, test_size=0.2)
    prior = get_prior(y_train)
    class_conditionals = get_class_conditionals(x_train, y_train)
    predictions = predict_class(prior, class_conditionals, x_test)
    accuracy = accuracy_score(y_test, predictions)  # подсчет точности
предсказаний

    print("4.1. Примеры предсказаний (0-setosa, 1-versicolor):\n")
    print("Для 1.0", predict_class(prior, class_conditionals, [[1.0]]))
    print("Для 1.9", predict_class(prior, class_conditionals, [[1.9]]))
    print("Для 5.1", predict_class(prior, class_conditionals, [[5.1]]))
    print("Для 6.0", predict_class(prior, class_conditionals, [[6.0]]))
    # 4.2
    print("Точность: {:.4f}".format(accuracy))
    print("Точность: 0.856\nТочность предсказания высокая поскольку входные
данные сильно отличаются\nдруг от друга"
          "и их легко классифицировать.\n")

def get_prior(y):  # априорное распределение вероятностей
```

```
    num_classes = tf.reduce_max(y) + 1
    probs = np.zeros((num_classes))
    for item in y:
        probs[item] += 1
    probs = probs/len(y)
    return tfd.Categorical(probs = probs)
```

Формулы для априорных вероятностей:

$$P(1)p(x|1) = P(2)p(x|2) \quad \text{или} \quad \ln\frac{p(x|1)}{p(x|2)} + \ln\frac{P(1)}{P(2)} = 0$$

$$\frac{(x-\mu_2)^2}{\sigma_2^2} - \frac{(x-\mu_1)^2}{\sigma_1^2} = 2\ln\left(\frac{P(2)}{P(1)}\frac{\sigma_1}{\sigma_2}\right)$$

```
def get_class_conditionals(x, y): # условное распределение
    num_classes = tf.reduce_max(y).numpy() + 1
    num_samples = x.shape[0]
    num_features = x.shape[1]
    loc = np.zeros((num_classes, num_features))
    scale_diag = np.zeros((num_classes, num_features))
    nos_in_class = np.zeros((num_classes,))

    # Подсчитываем объем
    for i in range(num_samples):
        nos_in_class[y[i]]+=1


    nos_in_class = np.expand_dims(nos_in_class, axis = 1)
    for i in range(num_samples):
        for j in range(num_features):
            loc[y[i], j] += x[i,j]

    # Мат ожидание
    loc = loc/nos_in_class
    for i in range(num_samples):
        for j in range(num_features):
            scale_diag[y[i], j] += (x[i,j]-loc[y[i],j])**2

    # Квадрат отклонений
    scale_diag = np.sqrt(scale_diag/nos_in_class)
    return tfd.MultivariateNormalDiag(loc = loc, scale_diag = scale_diag)


def predict_class(prior, class_conditionals, x): # вычисляет вероятности
классов, возвращает тот, у котрого он больше
    num_classes = tf.squeeze(class_conditionals.batch_shape)
    probs = []
    for i in range(num_classes):
        probs.append(tf.cast(class_conditionals[i].prob(x),dtype =
tf.float32)*tf.cast(prior.probs[i], dtype = tf.float32))
    predictions = np.argmax(probs, axis = 0)
```

```
    return predictions
```