

Федеральное государственное автономное образовательное учреждение
высшего образования

Национальный исследовательский технологический университет

«МИСиС»

Институт ИТАСУ

Кафедра Инженерной кибернетики

Лабораторная работа №1
по курсу «Нейронные сети и машинное обучение»

Выполнил:

Студент гр. МПИ-20-4-2

Малынковский О.В.

Проверил:

Курочкин И.И.

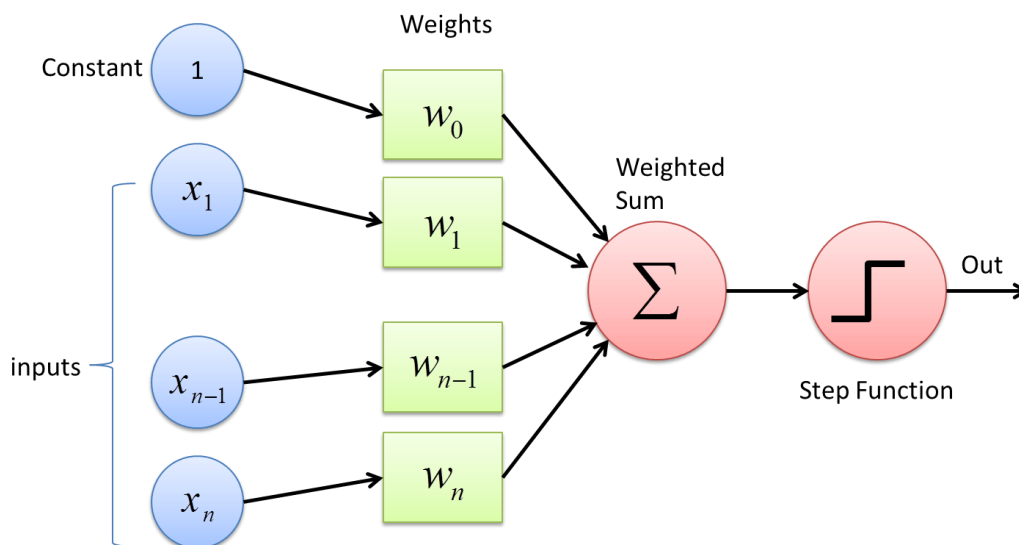
Москва 2020г.

Оглавление

Теория.....	3
Реализация.....	5
Примеры работы	6
Таблица сравнения методов на тестовых выборках	23
Вывод.....	24
Инструкция по запуску	25

Теория

Перцептрон представляет собой линейный классификатор, состоящий из следующих компонентов:



Аналогичное строение имеет и другой линейный классификатор – логистическая регрессия, за исключением того, что в качестве активационной функции берут не пороговую, а какую-нибудь дифференцируемую (например, sigmoid).

Однако если использовать только один нейрон, то классификация будет возможна только бинарная (хотя конечно можно было бы взять разделить отрезок от 0 до 1 на части в зависимости от количества классов, но у нас нет никаких предпосылок чтобы, например считать один класс «больше» другого).

В связи с этим необходимо взять сразу несколько таких однотипных нейронов (в классическом варианте столько же сколько и классов в данных), получив, по сути, уже однослойную нейронную сеть.

Для обучения однослойного перцептрона был использован метод коррекции ошибок, где веса обновляются по следующему правилу:

$$w_{new} = w_{old} + \delta * (y_{target} - y_{calculated}) * x$$

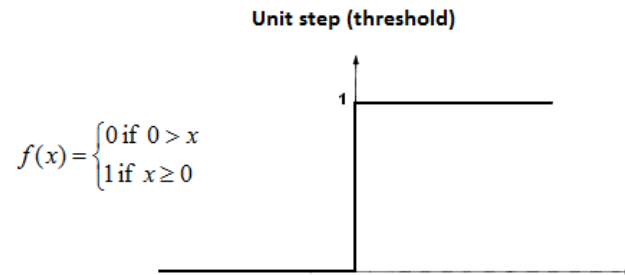
x -входные данные

δ -скорость обучения (от 0(не включая) до 1)

Значения, выдаваемые в качестве предсказания, будем считать по следующей формуле:

$$y_{target} = \sigma(\sum_i w_i * x_i + b_0)$$

σ -некоторая функция активации. Для перцептрона это пороговая:



Для логистической регрессии функция сигмоидальная(Sigmoid):

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

Она принимает значения между 0 и 1, поэтому мы можем интерпретировать вычисленные значения как вероятность принадлежности к некоторому классу.

Область значений этой функции включает значения всей числовой оси, поэтому, чтобы и в этом случае можно было ввести параллель с вероятностями, можно воспользоваться таким приёмом (soft_max):

У нас есть вектор $y_{target} = \{y_0, \dots, y_n\}$

Проекспоненцируем этот вектор $\{\exp(y_0), \dots, \exp(y_n)\}$ и поделим каждый элемент на суммы всех элементов полученного массива:

$$\left\{ \frac{\exp(y_0)}{\sum_i \exp(y_i)}, \dots, \frac{\exp(y_n)}{\sum_i \exp(y_i)} \right\}$$

Таким образом получили обобщение sigmoid на случай нескольких классов – softmax.

Обучение для логистической регрессии производится с помощью стохастического градиентного спуска:

$$w_{new} = w_{old} + \delta * \frac{dL}{dw}$$

L -функция ошибки.

Для обучения коррекция ошибок перцептрона не нужен градиент L, но в современных пакетах используется градиентный метод (отрицательный градиент указывает в сторону минимума, и мы к нему будем спускаться. А при коррекции ошибок метод сходится только в случае линейно-разделимых

данных). В связи с этим возьмем среднеквадратичную ошибку (и вместо пороговой функции активации возьмем сигмовидную для обеспечения дифференцируемости), так как её градиент, по сути, повторяет правило коррекции ошибок. Для логистической регрессии возьмем логистическую функцию потерь:

$$E = \frac{1}{2} \sum_{i=1}^N (y_{target} - y_{calculated})^2 - \text{среднеквадратичная (MSE)}$$
$$E = -\frac{1}{N} \sum_{i=1}^N (y_{target} * \log(y_{calculated}) + (1 - y_{target}) * \log(1 - y_{calculated}))$$

– логистическая (Log_loss)

Реализация

На языке Python с использованием библиотеки Keras реализуем обучение перцептрона. Добавляем в модель полно связный слой и активационную функцию. Параметр `learning_rate` задает скорость обучения, `loss` -функцию потерь, `optimizer` – метод обучения, далее указан список метрик. Для обучения задаем `epochs` – число эпох, `validation_split` – доля из обучающей выборки под валидационную часть.

```
model = Sequential()
model.add(Dense(6, input_shape=(X_train_std.shape[1],)))
model.add(Activation('softmax'))
sgd = SGD(learning_rate=1)
model.compile(loss='mse', optimizer=sgd, metrics=['accuracy'])
model.fit(X_train_std,y,epochs=50, validation_split=0.1, callbacks=[plot_losses])
```

Для получения модели логистической регрессии меняем функцию потерь на логистическую (`categorical_crossentropy`)

```
model = Sequential()
model.add(Dense(6, input_shape=(X_train_std.shape[1],)))
model.add(Activation('softmax'))
sgd = SGD(learning_rate=0.1)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
model.fit(X_train_std,y,epochs=100, validation_split=0.1, callbacks=[plot_losses])
```

Теперь посмотрим на эти модели в действии, применив к трем наборам данных.

Примеры работы

Датасет 1 - <https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression>

Mice Protein Expression Data Set

Данные представляют собой таблицу замеров уровней экспрессии 77 белков, измеренные в коре головного мозга 8 классов мышей (контрольных мышей и мышей с синдромом Дауна, подвергшихся условному условию контекстного страха). Итого таблица содержит 1080 строк.

Классы:

1. c-CS-s: контрольные мыши, стимулированные к обучению, которым вводили физиологический раствор (9 мышей)
2. c-CS-m: контрольные мыши, стимулированные к обучению, которым вводили мексалантин (10 мышей)
3. c-SC-s: контрольные мыши, не стимулированные к обучению, которым вводили физиологический раствор (9 мышей)
4. c-SC-m: контрольные мыши, не стимулированные к обучению, которым вводили мексалантин (10 мышей)
5. t-CS-s: мыши с трисомией, стимулированные к обучению, которым вводили физиологический раствор (7 мышей)
6. t-CS-m: мыши с трисомией, стимулированные к обучению, которым вводили мексалантин (9 мышей)
7. t-SC-s: мыши с трисомией, не стимулированные к обучению, которым вводили физиологический раствор (9 мышей)
8. t-SC-m: мыши с трисомией, не стимулированные к обучению, которым вводили мексалантин (9 мышей)

Проведем предобработку данных:

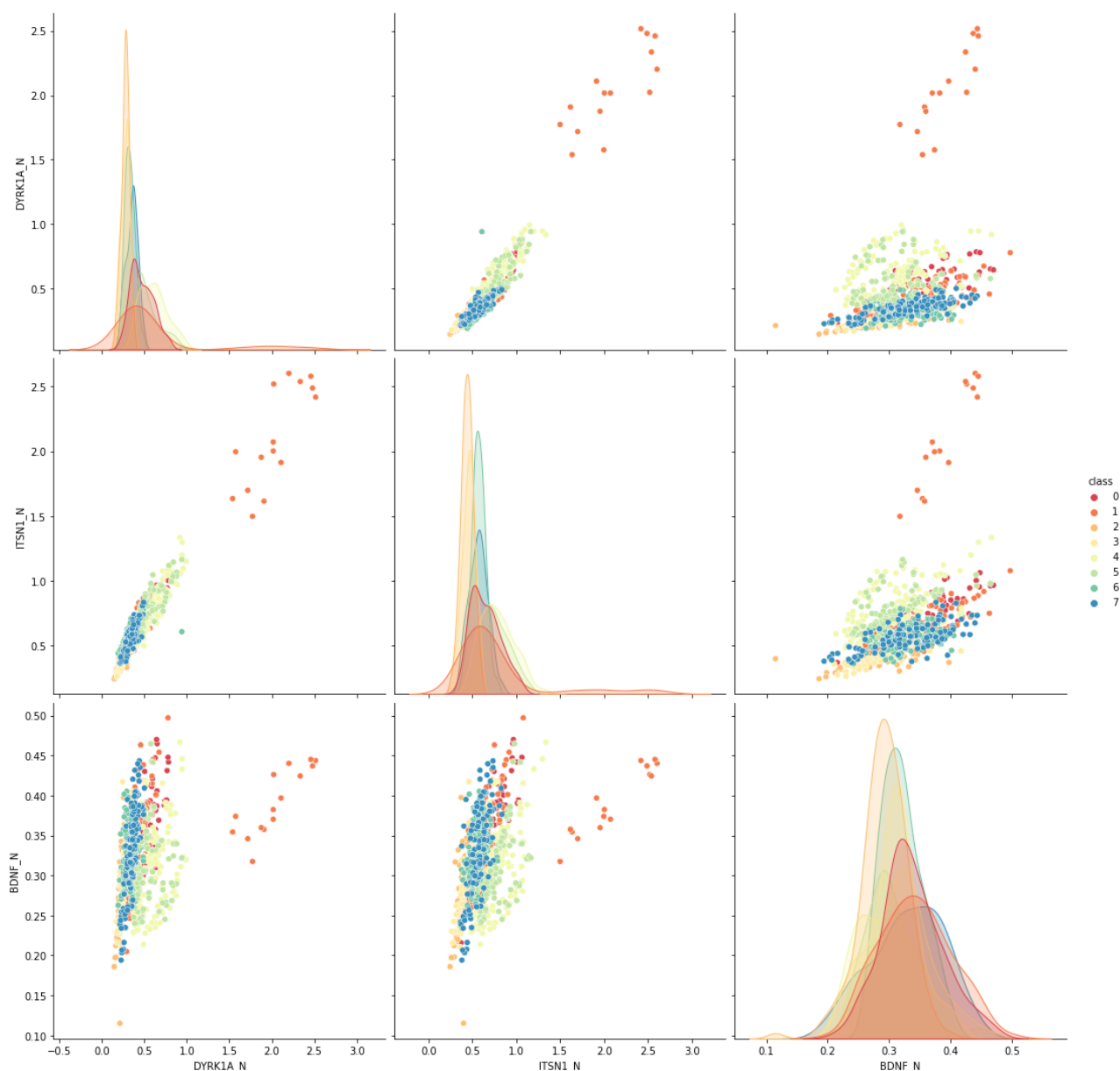
	MouseID	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	pBRAF_N	pCAMKII_N	pCREB_N	pELK_N	pERK_N	pJ
0	309_1	0.503644	0.747193	0.430175	2.816329	5.990152	0.218830	0.177565	2.373744	0.232224	1.750936	0.687906	0.30
1	309_2	0.514617	0.689064	0.411770	2.789514	5.685038	0.211636	0.172817	2.292150	0.226972	1.596377	0.695006	0.29

pCFOS_N	SYP_N	H3AcK18_N	EGR1_N	H3MeK4_N	CaNA_N	Genotype	Treatment	Behavior	class
0.108336	0.427099	0.114783	0.131790	0.128186	1.675652	Control	Memantine	C/S	c-CS-m
0.104315	0.441581	0.111974	0.135103	0.131119	1.743610	Control	Memantine	C/S	c-CS-m

Колонку MouseID можно удалить, так это просто нумерация для датасета. Так как MouseID нигде не повторяется, делаем вывод что здесь нет измерения для одной и той же мыши, т. е. этот признак можно отбросить.

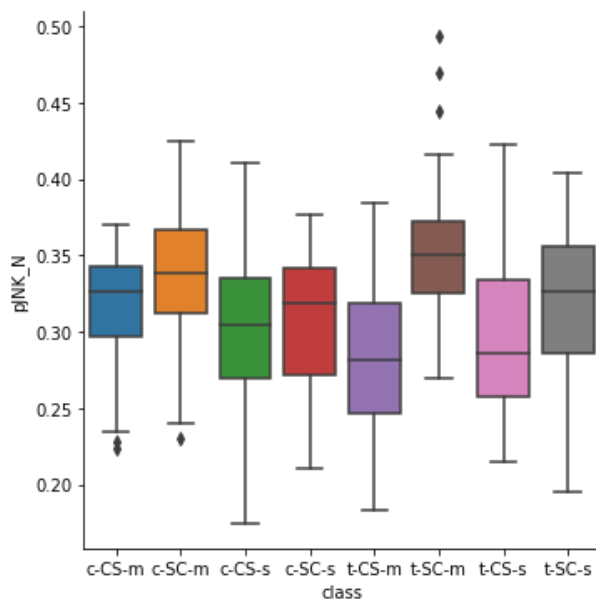
Есть 4 колонки с категориальными данными: 'Genotype', 'Treatment', 'Behavior', 'class'. Метки класса просто занумеруем с помощью LabelEncoder. Остальные три признака удалим, так как они составляют комбинации, представляющие классы.

В нашем датасете есть колонки (признаки), где относительно большое число пропусков (больше 180 при том, что всего 1080 строк). Это признаки: 'BAD_N', 'BCL2_N', 'H3AcK18_N', 'EGR1_N', 'H3MeK4_N'. Удалим эти колонки из нашего рассмотрения. В строках также обнаружены пропуски, их мы удалим (Заполнение их чем-то в данном случае трудно обосновать, но таких пропусков относительно мало).



На рисунке выше представлены графики рассеяния scatterplot для первых трех признаков. (Графики на диагонали показывают распределения значения трех первых белков, разделенные на классы цветом). Можно убедиться, что большинство классов являются линейно-неразделимыми.

Аналогично, можно убедиться в том, что классы линейно-неразделимы, построив boxplot (на рисунке пример для одного из 77 белков, цветом показаны классы мышей). Грубо говоря, если на графике медиана одного класса входит в область между 25ым и 75 процентилем (внутри ящика) и наоборот, то можно предположить, что две выборки не отличаются (однако надо провести тесты). Визуально видно, что несколько классов при таком подходе не сильно различаются, что подтверждает линейную неразделимость исходных данных.



Количество классов не совсем сбалансированно. Так как в целом различия не критичны, то искусственно увеличивать или уменьшать выборку не будем.

```
df["class"].value_counts()
```

2	150
6	135
4	120
3	120
1	120
0	120
7	117
5	90

Далее произведем стандартизацию (масштабирование данных для приведения к среднему в нуле и единичной дисперсии), а также разделим выборку на обучающую (например, 70%) и тестовую (30%). Также будем использовать валидационную выборку (10% от обучающей), но её выделение встроено в модель классификатора.

```

] feature_names = list(df.columns)
  feature_names.remove('class')
  X = df[feature_names]
  y = df['class']
  # Split the data into 70% training data and 30% test data
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
  # Train the scaler, which standarizes all the features to have mean=0 and
  sc = StandardScaler()
  sc.fit(X_train)
  # Apply the scaler to the X training data
  X_train_std = sc.transform(X_train)

  # Apply the SAME scaler to the X test data
  X_test_std = sc.transform(X_test)

```

Посредством OneHotEncoding закодируем метки классов.

```

encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform( np.array(y_train).reshape(-1, 1))

```

Данные предобработаны и можно запускать процесс обучения.

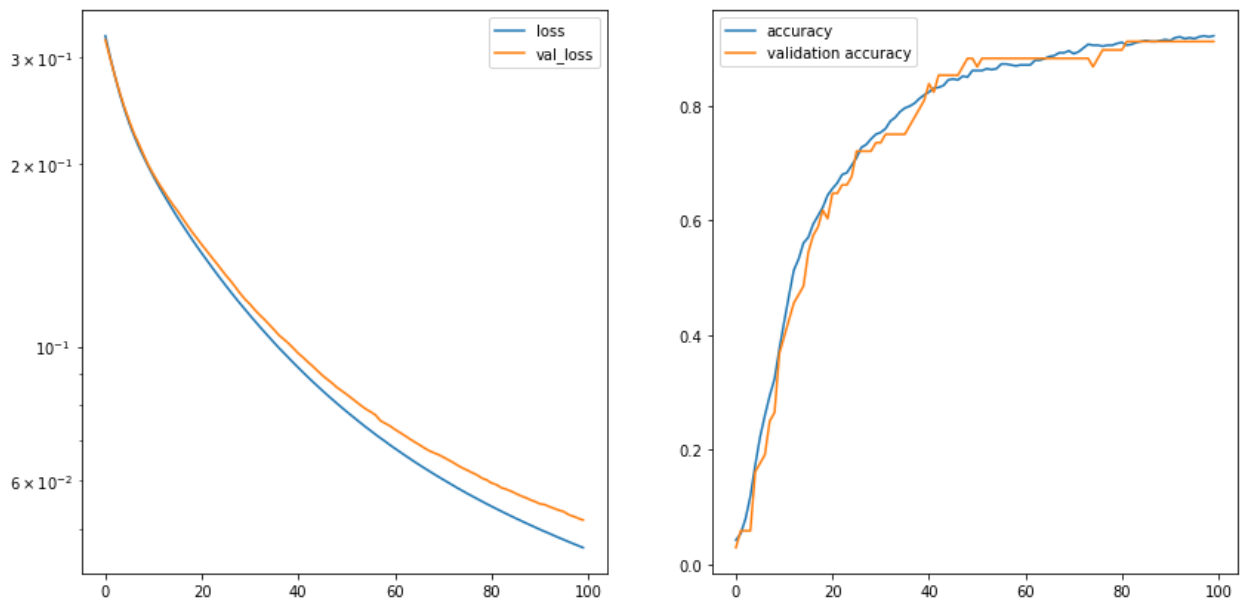
Перцептрон:

```

model = Sequential()
model.add(Dense(8, input_shape=(X_train_std.shape[1],)))
model.add(Activation(hard_sigmoid))
sgd = SGD(learning_rate=0.1)
model.compile(loss='mse', optimizer=sgd, metrics=['accuracy'])
model.fit(X_train_std,y,epochs=100, validation_split=0.1, callbacks=[plot_losses])

```

На следующем рисунке (и далее в работе на аналогичных рисунках) слева график ошибки, а справа ассигасу. По горизонтальной оси номера эпох обучения. Синяя линия показывает результаты для обучающей выборки, а оранжевая – для валидационной.



loss: 0.0466 - accuracy: 0.9216 - val_loss: 0.0517 - val_accuracy: 0.9118

Ассурасу на обучающей выборке достиг 0.92. Более подробные результаты покажем для тестовой выборки:

	precision	recall	f1-score	support
0	0.76	0.80	0.78	35
1	0.91	0.71	0.80	45
2	0.98	0.92	0.95	48
3	1.00	1.00	1.00	32
4	0.65	0.89	0.75	37
5	0.96	0.73	0.83	30
6	0.94	0.97	0.95	32
7	0.92	1.00	0.96	33
accuracy			0.87	292
macro avg	0.89	0.88	0.88	292
weighted avg	0.89	0.87	0.87	292

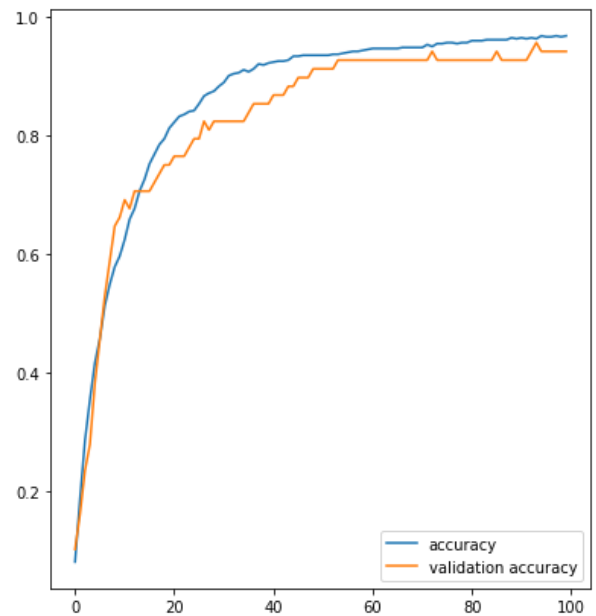
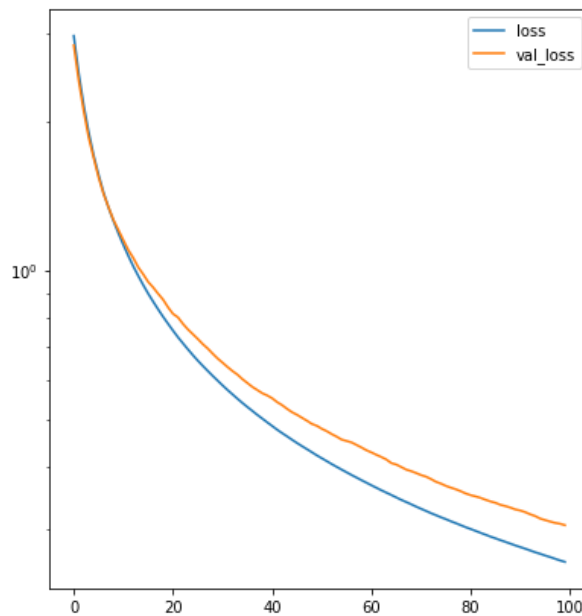
Слева цифрами указаны классы. Лишь один из них идеально предсказан моделью (это можно объяснить, если взглянуть на приведенный ранее scatter plot, где был один класс точки которого были легко отделимы от других линияй). В среднем на тестовой выборке ассурасу составил 0.87, что в целом неплохо для такой простой модели.

Логистическая регрессия:

```

model = Sequential()
model.add(Dense(8, input_shape=(X_train_std.shape[1],)))
model.add(Activation('softmax'))
sgd = SGD(learning_rate=0.1)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
model.fit(X_train_std,y,epochs=100, validation_split=0.1, callbacks=[plot_losses])

```



loss: 0.2571 - accuracy: 0.9673 - val loss: 0.3051 - val accuracy: 0.9412

```

y_pred = model.predict_classes(X_test_std)
print(classification_report(y_test, y_pred))

```

	precision	recall	f1-score	support
0	0.69	0.84	0.76	32
1	0.86	0.77	0.81	39
2	0.92	0.98	0.95	46
3	1.00	0.98	0.99	46
4	0.89	0.86	0.88	37
5	1.00	0.91	0.95	23
6	1.00	0.93	0.96	40
7	0.97	1.00	0.98	29
accuracy			0.91	292
macro avg	0.92	0.91	0.91	292
weighted avg	0.92	0.91	0.91	292

В случае логистической регрессии результат чуть лучше, для тестовой выборки accuracy уже 0.91. В целом обе модели ожидаемо демонстрируют приблизительно одинаковый результат.

Датасет 2 - <http://archive.ics.uci.edu/ml/datasets/Abalone>

Abalon Data Set

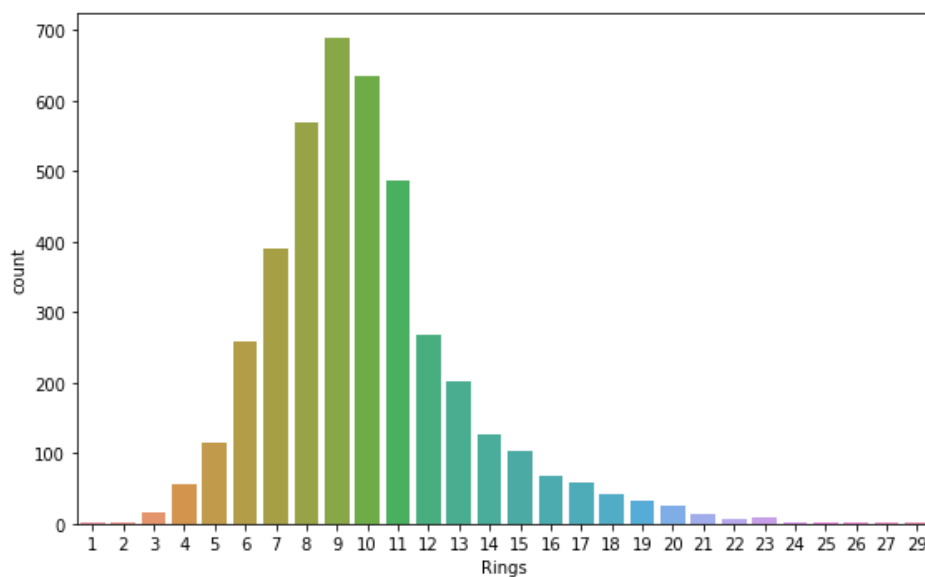
Данные представляют собой таблицу измерений 8 физических параметров морских ракушек. По этим данным нужно предсказать возраст ракушки, который считается по числу колец на срезе. Возраст (число колец) здесь является классом. Итого таблица содержит 4417 строк.

Признаки:

- Sex (категориальный) - M, F, and I (infant)
- Length / Longest shell measurement (числовой)
- Diameter / perpendicular to length (числовой)
- Height / with meat in shell (числовой)
- Whole weight (числовой)
- Shucked weight (числовой)
- Viscera weight (числовой)
- Shell weight (числовой)

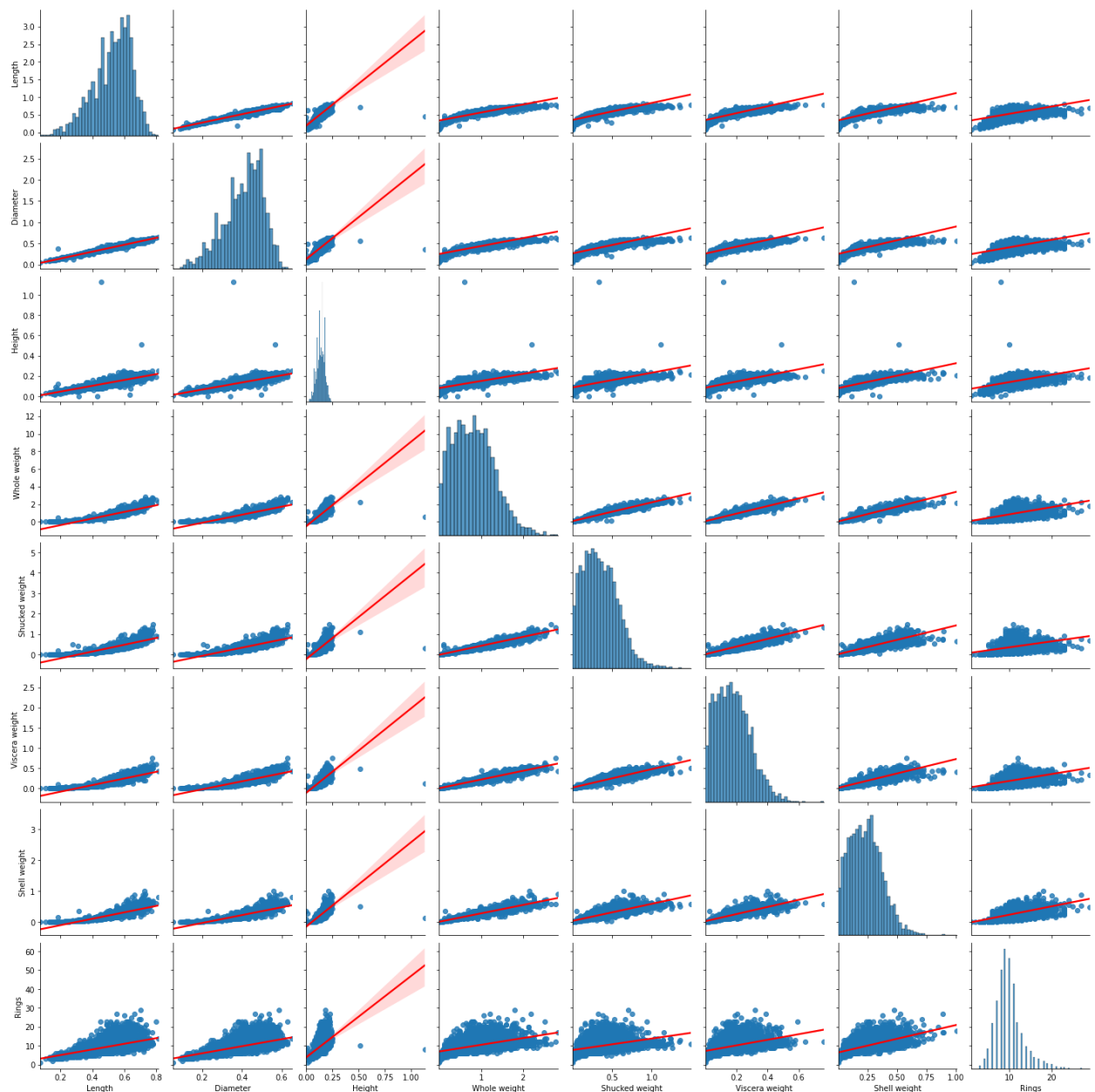
Проведем предобработку данных:

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

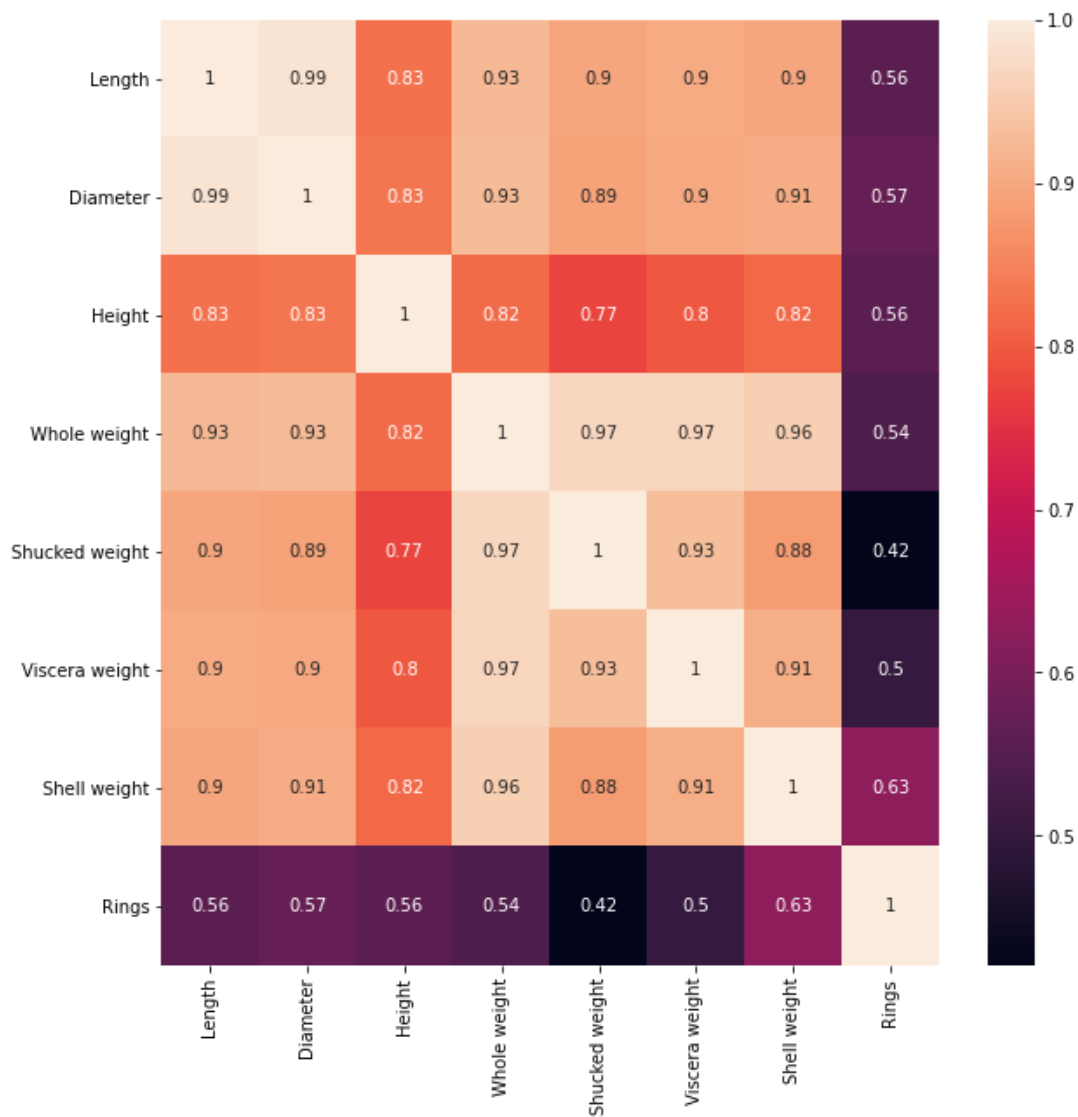


В датасете Abalone нет пропущенных значений, но классы крайне не сбалансированы (пока не будем с этим ничего делать). Ракушек возрастом до 5 и от 16 и более крайне мало.

Также, как и для предыдущего датасета построим парные scatter plot (красной линией показана линейная аппроксимация, а светло-красном область вокруг прямой в границах которой возможна ошибка, широкий коридор возникает в областях где мало точек для точной подстройки коэффициентов прямой).



Большая часть признаков распределены нормально или по распределению Стюдента. Интересно, что все признаки находятся почти в прямолинейной зависимости друг с другом (но это нарушается в случае с количеством колец – в нижней строке или самом правом столбце). Матрица корреляции дополнительно подтверждает линейную зависимость:



Далее с помощью oneHotEncoding закодируем первый признак, затем проведем стандартизацию данных также как и для предыдущего датасета.

```
df2 = pd.get_dummies(df2, columns=['Sex'])
feature_names = list(df2.columns)
feature_names.remove('Rings')
X = df2[feature_names]
y = df2['Rings']
# Split the data into 70% training data and 30% test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# Train the scaler, which standarizes all the features to have mean=0 and
sc = StandardScaler()
sc.fit(X_train)
# Apply the scaler to the X training data
X_train_std = sc.transform(X_train)

# Apply the SAME scaler to the X test data
X_test_std = sc.transform(X_test)
```

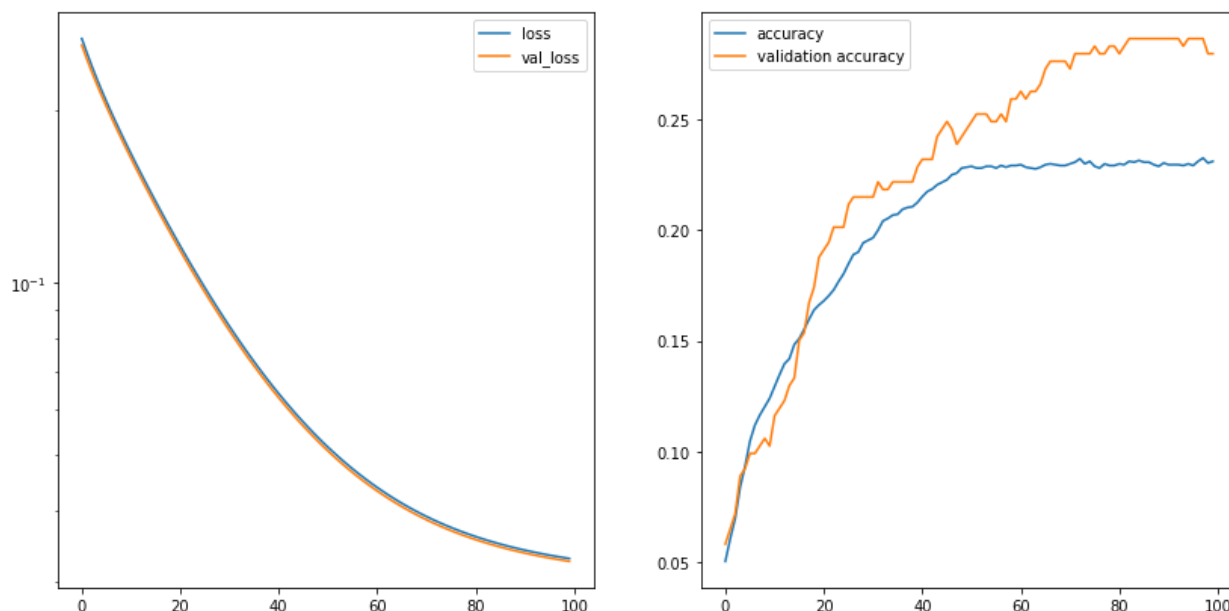
Можем провести обучение модели.

Перцептрон:


```

encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform( np.array(y_train).reshape(-1, 1))
model = Sequential()
model.add(Dense(28, input_shape=(X_train_std.shape[1],)))
model.add(Activation(hard_sigmoid))
sgd = SGD(learning_rate=0.1)
model.compile(loss='mse', optimizer=sgd, metrics=['accuracy'])
model.fit(X_train_std,y,epochs=100, validation_split=0.1, callbacks=[plot_losses])

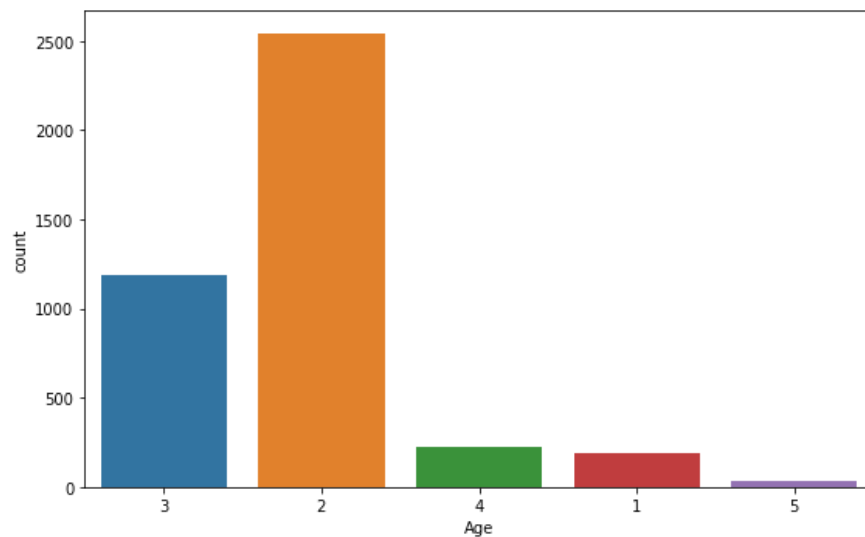
```



loss: 0.0330 - accuracy: 0.2312 - val_loss: 0.0326 - val_accuracy: 0.2799

Результат ассигасу меньше 0.3 никак не может удовлетворить. Вероятно, такой результат связан с очень большой несбалансированностью классов и для решения задачи больше подошла регрессия с округлением выхода модели для получения целочисленного возраста ракушек.

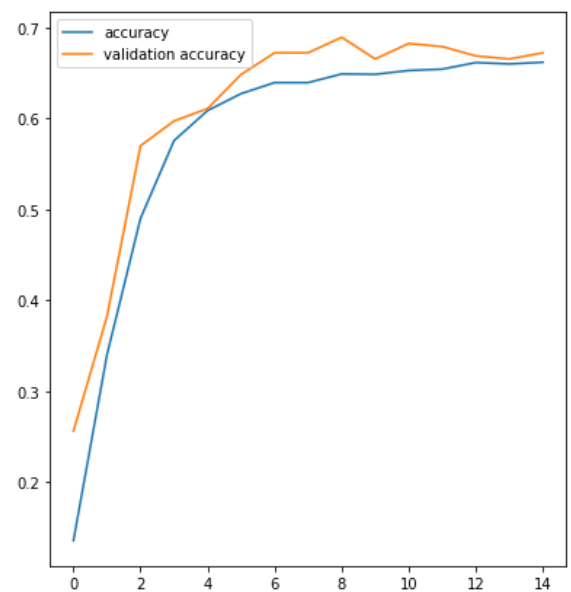
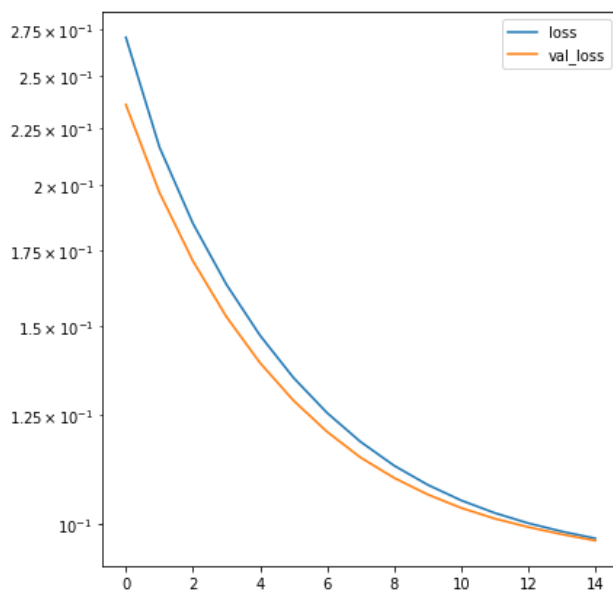
Попробуем все-таки и с нашей моделью получить приемлемый результат. Искусственно уменьшить или увеличить выборку по классам здесь нельзя, так как для некоторых классов больше 500 объектов в выборке, а для некоторых единицы. Попробуем уменьшить число классов, разделив все возраста на 5 групп – 1 группа (возраст от 0 до 5), 2 группа (6-10), 3 группа (11-15), 4 группа (16-20), 5 группа (остальные)



После такого изменения в одном из классов значений относительно мало, но хотя бы уже больше 100. Попробуем построить теперь модель.

Перцептрон:

```
encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform( np.array(y_train).reshape(-1, 1))
model = Sequential()
model.add(Dense(5, input_shape=(X_train_std.shape[1],)))
model.add(Activation(hard_sigmoid))
sgd = SGD(learning_rate=0.1)
model.compile(loss='mse', optimizer=sgd, metrics=['accuracy'])
model.fit(X_train_std,y,epochs=15, validation_split=0.1, callbacks=[plot_losses])
```



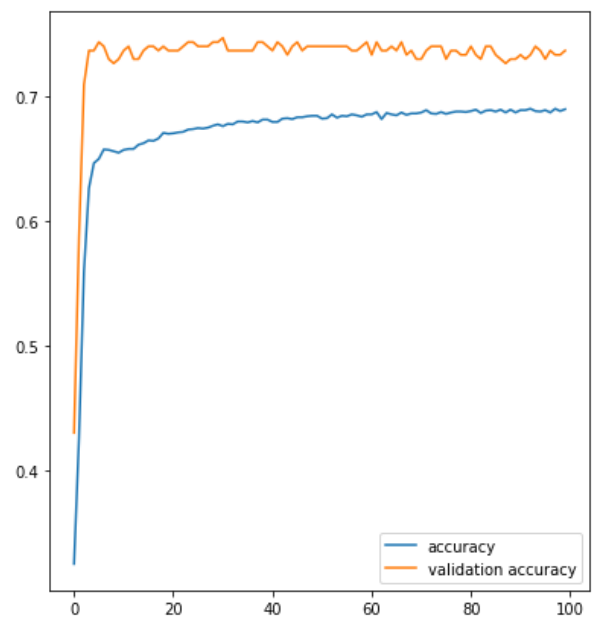
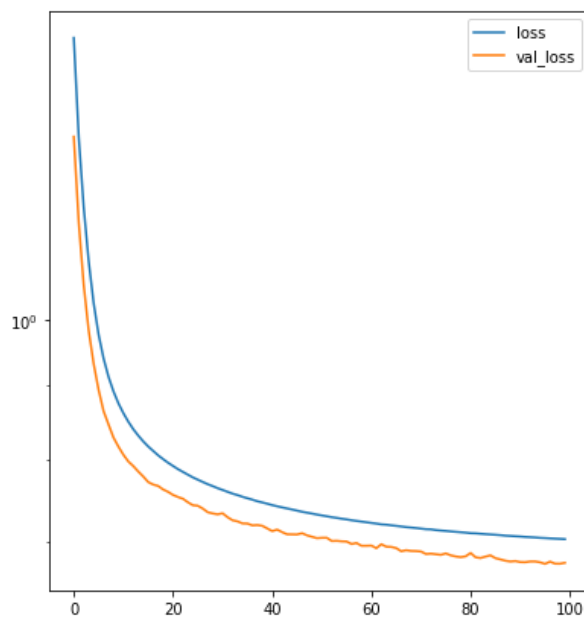
loss: 0.0972 - accuracy: 0.6620 - val_loss: 0.0967 - val_accuracy: 0.6724

```
y_pred = np.argmax(model.predict(X_test_std), axis=-1)+
print(classification_report(y_test, y_pred,zero_divisio
```

	precision	recall	f1-score	support
1	0.00	0.00	0.00	57
2	0.67	0.91	0.77	743
3	0.55	0.36	0.43	369
4	0.00	0.00	0.00	70
5	0.00	0.00	0.00	15
accuracy			0.65	1254
macro avg	0.24	0.25	0.24	1254
weighted avg	0.56	0.65	0.58	1254

Логистическая регрессия:

```
encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform( np.array(y_train).reshape(-1, 1))
model = Sequential()
model.add(Dense(5, input_shape=(X_train_std.shape[1],)))
model.add(Activation('softmax'))
sgd = SGD(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
model.fit(X_train_std,y,epochs=100, validation_split=0.1, callbacks=[plot_losses])
```



```
loss: 0.7033 - accuracy: 0.6901 - val_loss: 0.6770 - val accuracy: 0.7372
```

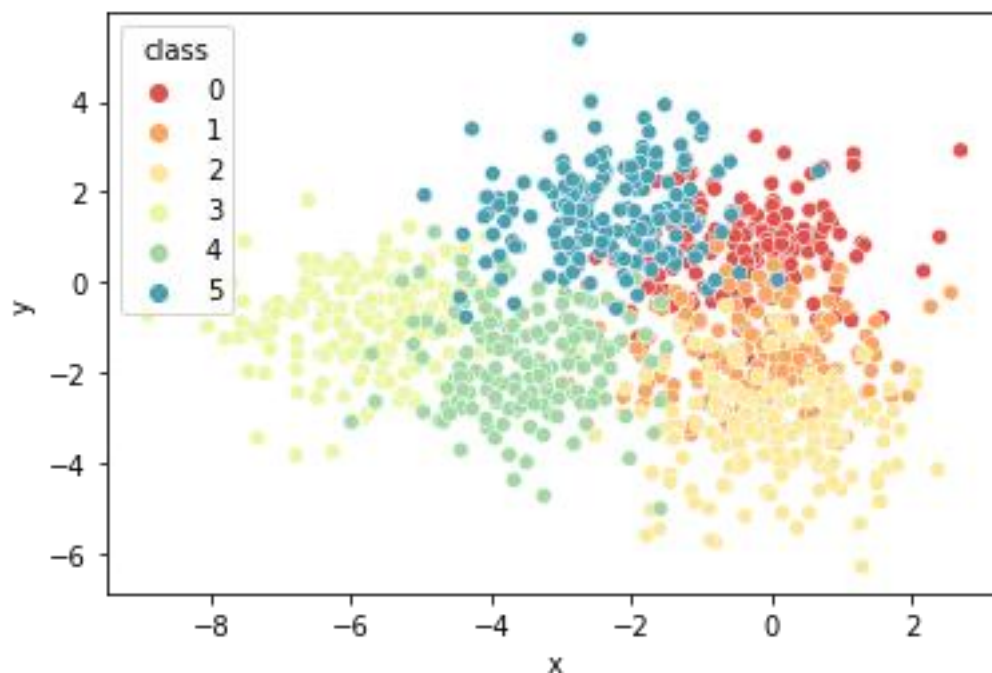
```
y_pred = np.argmax(model.predict(X_test_std), axis=-1)+1
print(classification_report(y_test, y_pred, zero_division=0))
```

	precision	recall	f1-score	support
1	0.80	0.60	0.69	62
2	0.73	0.88	0.80	753
3	0.52	0.43	0.47	350
4	0.50	0.07	0.12	74
5	0.00	0.00	0.00	15
accuracy			0.68	1254
macro avg	0.51	0.39	0.41	1254
weighted avg	0.65	0.68	0.65	1254

И перцептрон, и логистическая регрессия показали ожидаемо схожие результаты: ассигасу 0.65 и 0.68 на тестовой выборке соответственно. Но, к сожалению, самый малочисленный класс обе модели совсем не смогли обучиться предсказывать.

Датасет 3 – Сгенерирован самостоятельно, с двумя признаками для визуализации

Датасет состоит из двух координат x,y и метки класса (всего 6 классов). Выборка состоит из 1000 точек.



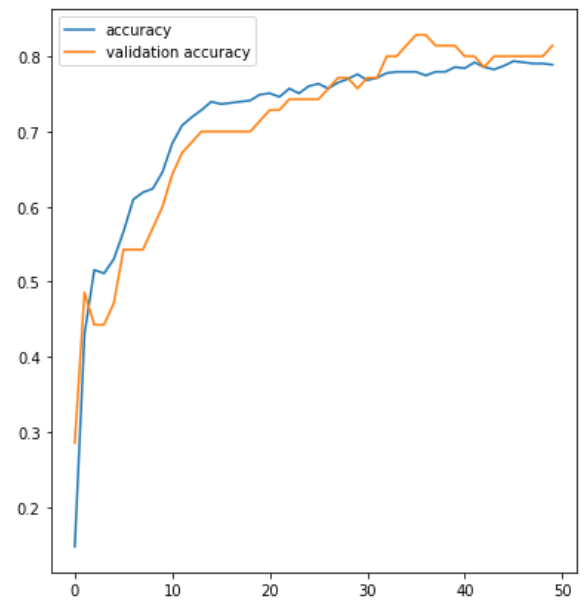
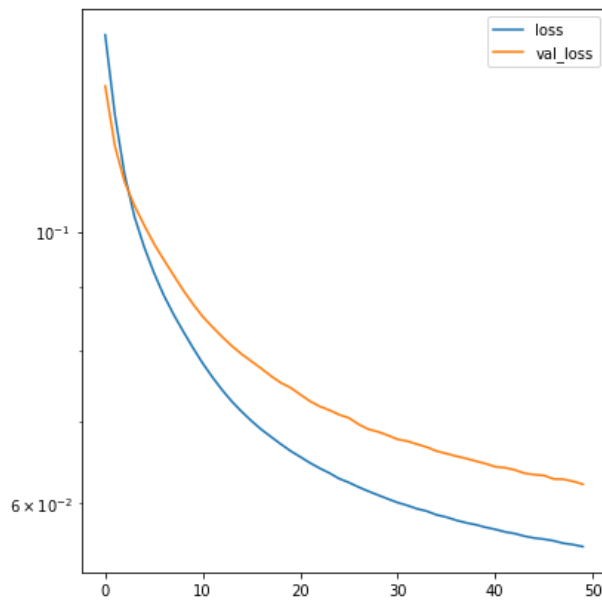
Так как данные сгенерированы мной, то пропусков в них нет, можно сразу провести стандартизацию.

```
df3 = pd.read_table('6class_30percent.txt',header=None,names=['x','y','class'], delimiter=' ')
X=df3[['x','y']]
y=df3['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

Данные готовы для обучения модели.

Перцептрон:

```
encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform( np.array(y_train).reshape(-1, 1))
model = Sequential()
model.add(Dense(6, input_shape=(X_train_std.shape[1],)))
model.add(Activation(hard_sigmoid))
sgd = SGD(learning_rate=1)
model.compile(loss='mse', optimizer=sgd, metrics=['accuracy'])
model.fit(X_train_std,y,epochs=100, validation_split=0.1, callbacks=[plot_losses])
```

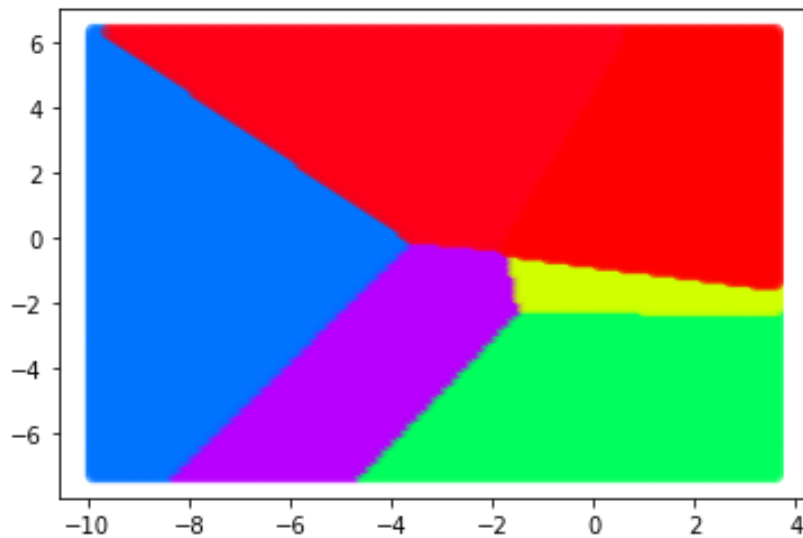


loss: 0.0553 - accuracy: 0.7889 - val_loss: 0.0622 - val_accuracy: 0.8143

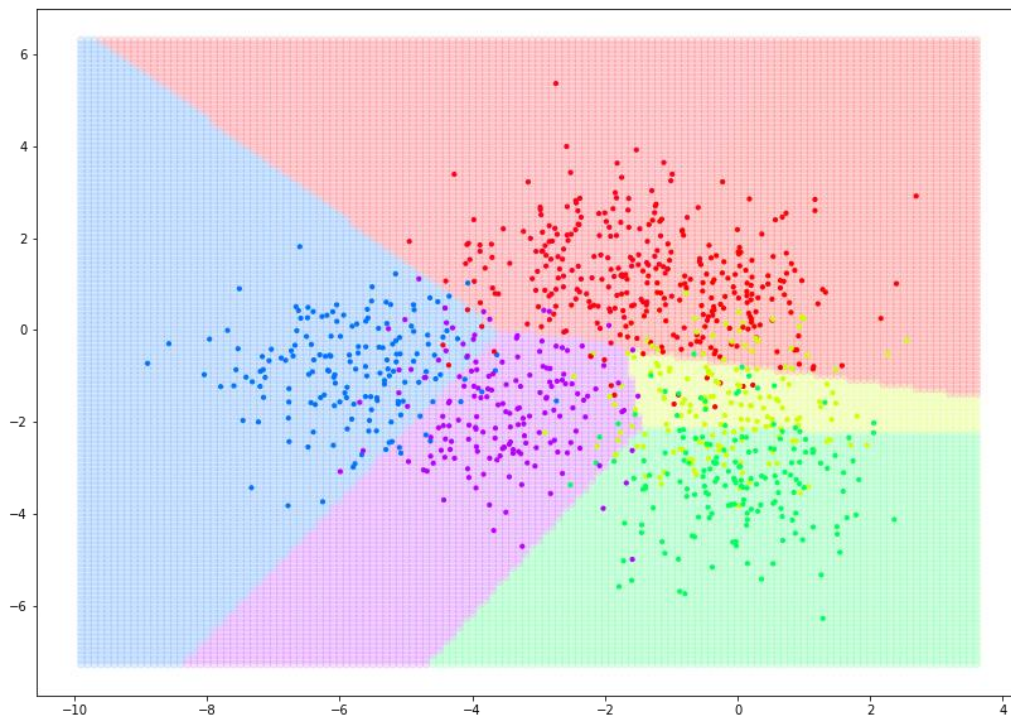
```
y_pred = np.argmax(model.predict(X_test_std), axis=-1)
print(classification_report(y_test, y_pred,zero_division=0))
```

	precision	recall	f1-score	support
0	0.64	0.76	0.69	46
1	0.77	0.40	0.53	67
2	0.68	0.91	0.78	54
3	0.77	0.92	0.84	37
4	0.77	0.73	0.75	41
5	0.87	0.87	0.87	55
accuracy			0.74	300
macro avg	0.75	0.77	0.74	300
weighted avg	0.75	0.74	0.73	300

Границы решений:

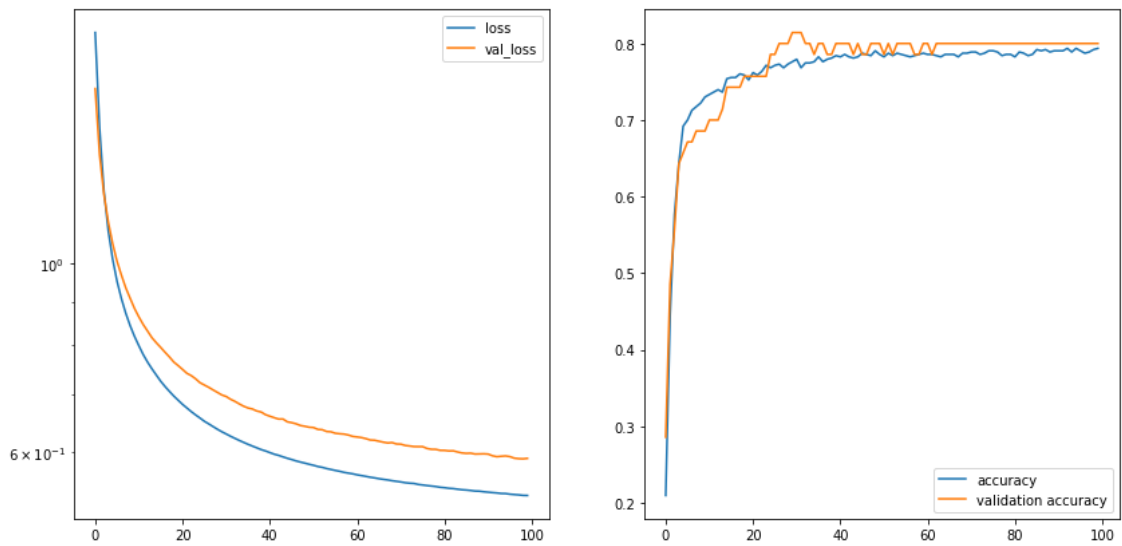


Граница решений + точки исходной выборки:



Логистическая регрессия:

```
model = Sequential()  
model.add(Dense(6, input_shape=(X_train_std.shape[1],)))  
model.add(Activation('softmax'))  
sgd = SGD(learning_rate=0.1)  
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])  
model.fit(X_train_std,y,epochs=100, validation_split=0.1, callbacks=[plot_losses])
```

loss: 0.5338 - accuracy: 0.7937 - val_loss: 0.5900 - val_accuracy: 0.8000

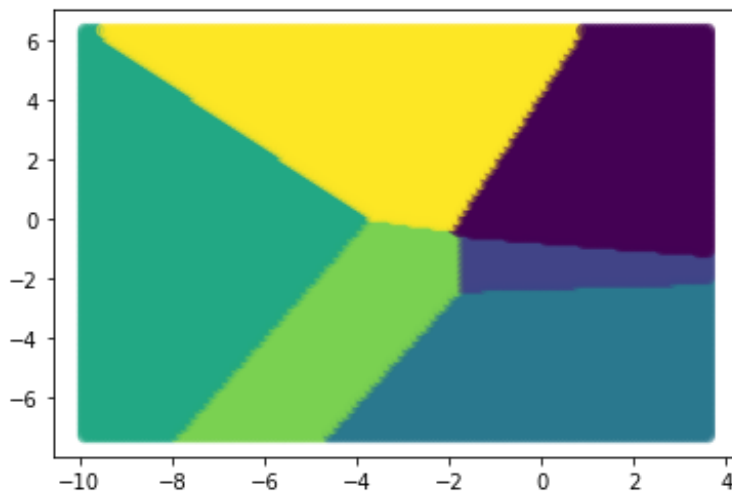
```

y_pred = np.argmax(model.predict(X_test_std), axis=-1)
print(classification_report(y_test, y_pred, zero_division=0))

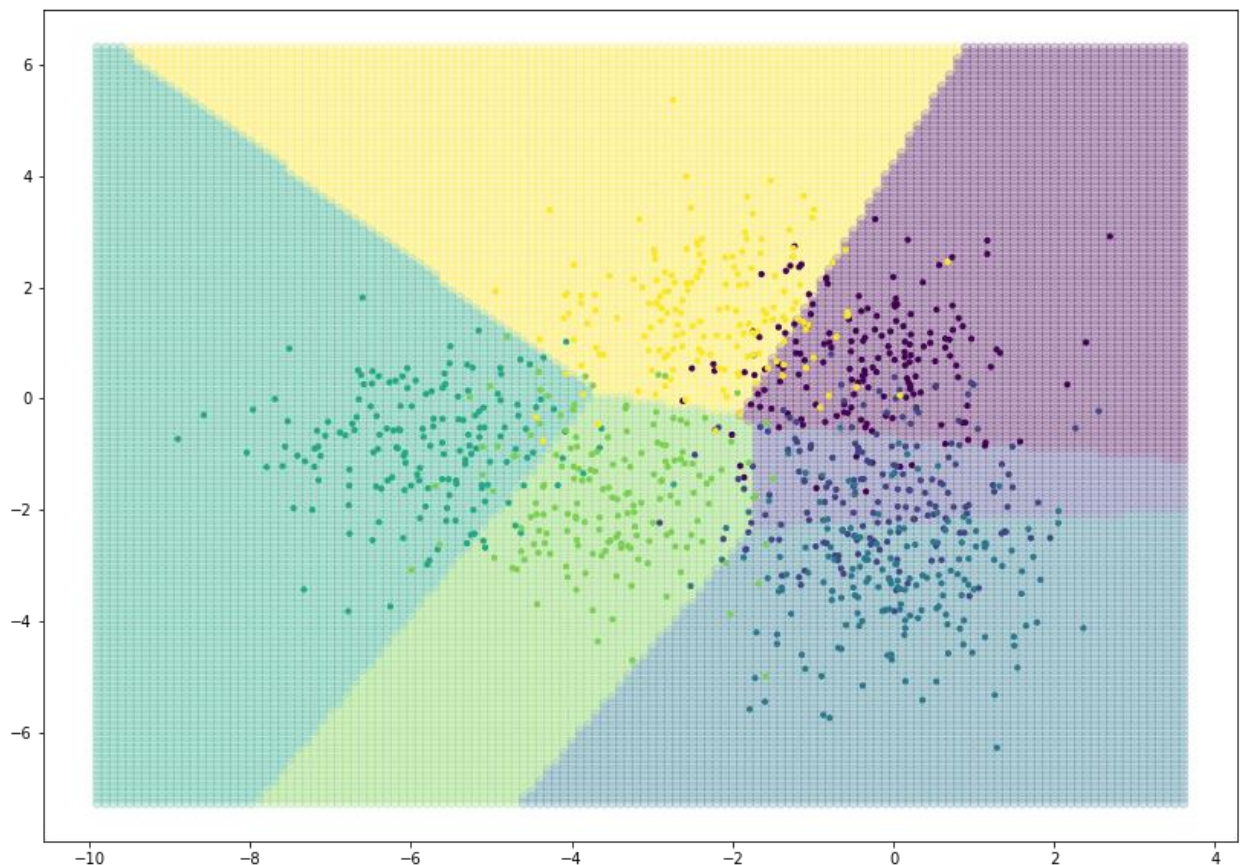
```

	precision	recall	f1-score	support
0	0.66	0.76	0.71	46
1	0.77	0.49	0.60	67
2	0.70	0.89	0.78	54
3	0.77	0.92	0.84	37
4	0.81	0.71	0.75	41
5	0.87	0.87	0.87	55
accuracy			0.76	300
macro avg	0.76	0.77	0.76	300
weighted avg	0.76	0.76	0.75	300

Границы решений:



Границы решений + точки исходной выборки:



И перцептрон, и логистическая регрессия показали ожидаемо схожие результаты: ассигасу 0.74 и 0.76 на тестовой выборке соответственно. Хуже всего в обоих случаях происходит выделение желтых точек на картинке для перцептрона и синих на картинке для логистической регрессии (цвета разные, но это один и тот же класс), (также у них для этого класса самый низкий recall, т.е. низкий процент объектов этого класса определен верно, но в целом высокий recall, т.е. нейрон отвечающий за этот класс часто дает ложные положительные срабатывания). Это связано с тем, что точки этого класса имеют достаточно большое количество пересечений с областями соседних классов, что затрудняет построение прямых, разделяющих их с высоким ассигасу.

Таблица сравнения методов на тестовых выборках

Precision, Recall, F1 -measure в таблицах ниже являются средневзвешенными значениями метрик. Их значения по отдельным классам были приведены ранее.

Датасет 1

	Accuracy	Precision	Recall	F1-measure
Однослойный перцептрон	0.87	0.89	0.87	0.87
Логистическая регрессия	0.91	0.92	0.91	0.91

Датасет 2

	Accuracy	Precision	Recall	F1-measure
Однослойный перцептрон	0.65	0.56	0.65	0.58
Логистическая регрессия	0.68	0.65	0.68	0.65

Датасет 3

	Accuracy	Precision	Recall	F1-measure
Однослойный перцептрон	0.74	0.75	0.74	0.73
Логистическая регрессия	0.76	0.76	0.76	0.75

Вывод

В ходе выполнения лабораторной работы были рассмотрены такие модели классификации как однослойный перцептрон и логистическая регрессия.

В результате использования этих моделей для трех датасетов и вычисления 4х метрик качества было установлено, что оба метода дают приблизительно одинаковый результат. Показатели качества у логистической регрессии слегка выше, но это лишь результат для одного из всевозможных наборов гиперпараметров для обучения.

Результат был ожидаем и лишь подтвержден на практике. Дело в том, что оба метода по сути имеют одинаковое строение. Несмотря на разные функции активации, и там и там происходит суммирование взвешенных ходов. В процессе обучения в обоих случаях строится гиперплоскость, отделяющая один класс от других, поэтому если использовать один и тот же датасет, то

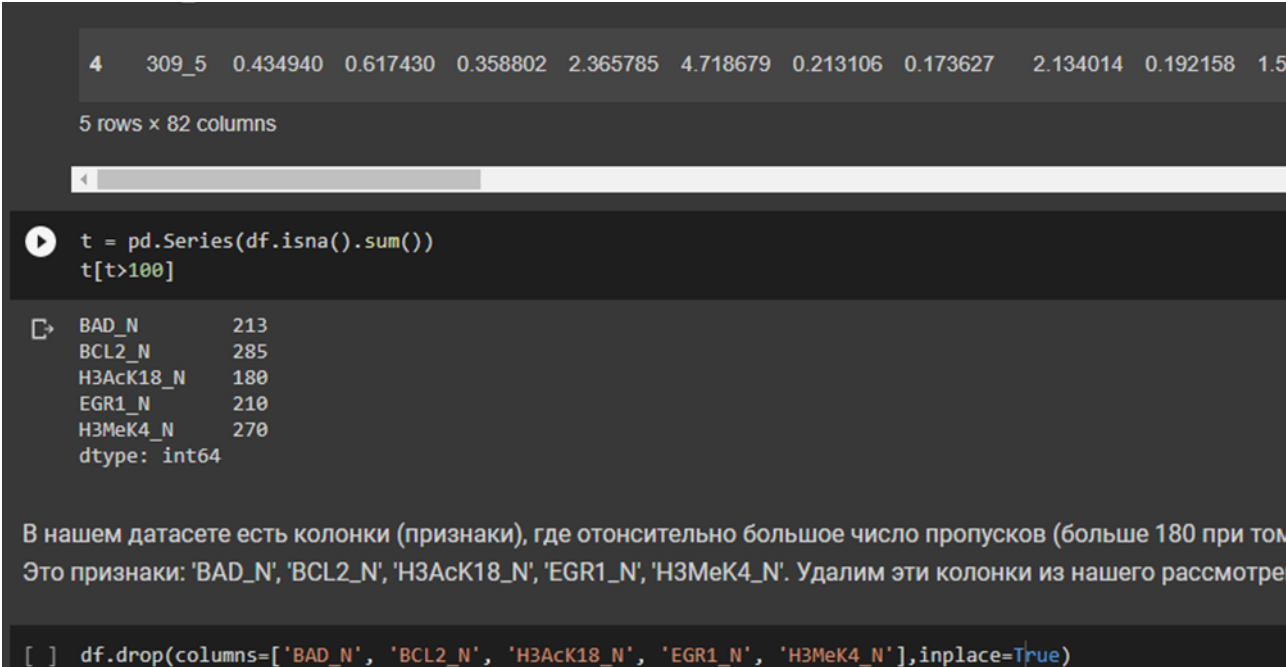
искомые гиперплоскости будет одни и те же, поэтому обе модели давали схожие результаты на выходе.

Главным отличием перцептрона от логистической регрессии является способ обучения: в первом случае методом коррекции ошибок, во втором методом стохастического градиента. Однако второй вариант предпочтительнее. В случае метода коррекции ошибок сходимость гарантирована только для линейно-разделимых данных.

Инструкция по запуску

Для запуска потребуется использовать Anaconda (Jupyter Notebook), но предпочтительнее запускать из Google Colab (<https://colab.research.google.com/drive/1GcKIEjfFNuCrFKUXLNDyOxx4crI0x8uH?usp=sharing>), так как там уже гарантированно установлены используемые библиотеки. Также все датасеты, использованные в отчете, должны лежать в одной папке с кодом, иначе нужно будет изменить в коде путь к ним.

Затем нужно просто последовательно выполнить код в ячейках. Никакие параметры менять не нужно, иначе не получится воспроизвести достигнутые результаты классификации в точности как полученные в отчете к работе.



```
4 309_5 0.434940 0.617430 0.358802 2.365785 4.718679 0.213106 0.173627 2.134014 0.192158 1.5
```

5 rows x 82 columns

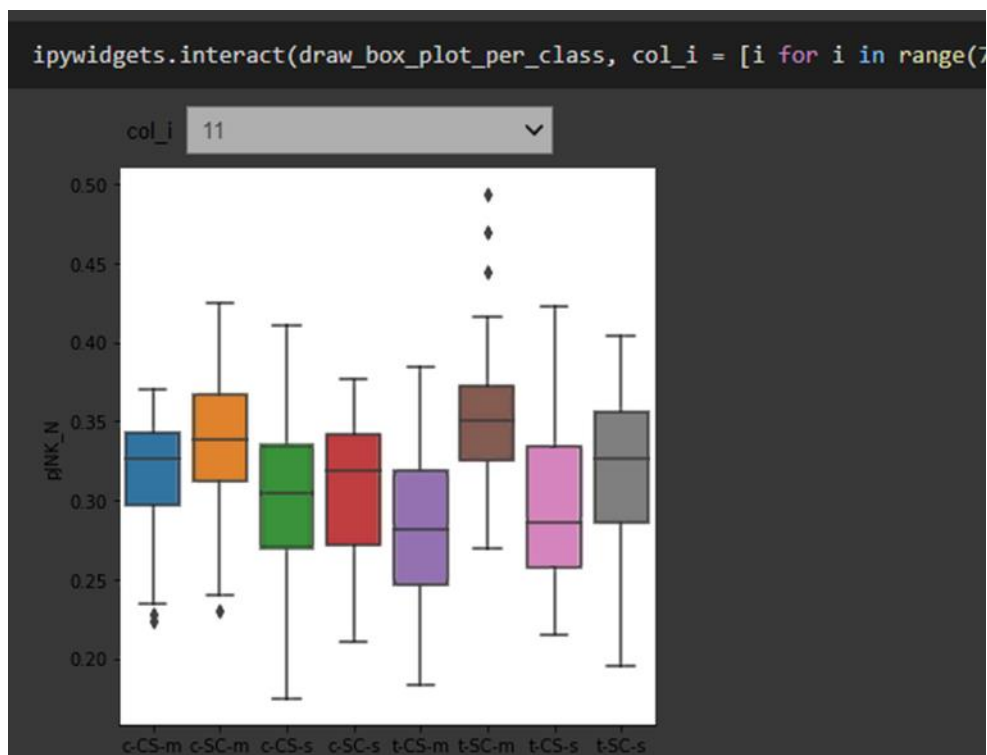
```
t = pd.Series(df.isna().sum())
t[t>100]
```

```
BAD_N      213
BCL2_N     285
H3AcK18_N  180
EGR1_N     210
H3MeK4_N   270
dtype: int64
```

В нашем датасете есть колонки (признаки), где отонсительно большое число пропусков (больше 180 при том
Это признаки: 'BAD_N', 'BCL2_N', 'H3AcK18_N', 'EGR1_N', 'H3MeK4_N'. Удалим эти колонки из нашего рассмотре

```
[ ] df.drop(columns=['BAD_N', 'BCL2_N', 'H3AcK18_N', 'EGR1_N', 'H3MeK4_N'],inplace=True)
```

Также в одной из ячеек есть возможность интерактивной постройки boxplot для первого датасета:



Для этого нужно в кнопке col_i выбрать номер (означает номер белка из исходной таблицы) и график перестроится