ПРАКТИЧЕСКАЯ РАБОТА №1. ФАКТОРНЫЙ АНАЛИЗ

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Факторный анализ — совокупность методов многомерного статистического анализа, применяемых для изучения взаимосвязей между значениями переменных. Цели факторного анализа:

- сокращение числа переменных;
- определение взаимосвязей между переменными, их классификация.

Корреляционная матрица – это квадратная таблица, в которой на пересечении строк и столбцов располагаются коэффициенты корреляции между соответствующими параметрами.

Методики факторного анализа:

1. Анализ главных компонент.

После того, как мы получили несколько групп переменных, объединенных одним фактором, мы можем повторить процедуру регрессии данных. Таким образом, факторы последовательно выделяются один за другим. Так как, каждый последующий фактор, определяется так, чтобы максимизировать изменчивость, оставшуюся от предыдущих, то факторы оказываются независимыми друг от друга. Другими словами, некоррелированными или ортогональными.

Существует два критерия для вычисления количества главных компонентов:

- критерий Кайзера;
- критерий «каменистой осыпи».

2. Анализ главных факторов

В анализе главных компонент предполагается, что должна быть использована вся изменчивость переменных, тогда как в анализе главных факторов вы используете только изменчивость переменной, общую и для других переменных. В большинстве случаев эти два метода приводят к весьма близким результатам. Однако анализ главных компонент часто более предпочтителен как метод сокращения данных, в то время как анализ главных факторов лучше применять с целью определения структуры данных.

3. Факторные нагрузки.

Факторными нагрузками называются корреляции между переменными и несколькими факторами (или «новыми» переменными, которые выделены по умолчанию).

4. Вращение факторной структуры.

Можно изобразить факторные нагрузки в виде диаграммы рассеяния. На такой диаграмме каждая переменная представлена точкой, а её координаты равны факторным нагрузкам. Можно повернуть оси в любом направлении без изменения относительного положения точек; однако действительные координаты точек, то есть факторные нагрузки, должны, без сомнения, меняться. Если поворачивать оси относительно начала координат, то можно достичь ясного представления о нагрузках, определяющих группы переменных.

Типичными методами вращения являются стратегии варимакс, квартимакс, и эквимакс.

5. Косоугольные факторы.

6. Иерархический факторный анализ.

ЦЕЛЬ РАБОТЫ

Основной задачей практической работы является выделение наиболее показательных системных счётчиков, которые косвенно могут давать нам информацию об остальных параметрах системы.

ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ И ПОРЯДОК ВЫПОЛНЕНИЯ

Для получения исходных данных необходимо:

1. Подобрать подходящие для анализа данные из репозитория, предоставляемого преподавателем.

2. Создать таблицу данных в MS Excel.

Обработка исходных данных:

Необходимо заменить не числовые данные. Для этого вводится любая шкала соответствия, на выбор студента, с обязательным внесением в отчет легенды по шкале.

3. Следует удалить из анализа переменные, дисперсия которых равна нулю.

4. Постройте корреляционную матрицу и сделайте соответствующие выводы.

5. Откройте программу STATISTICA. Создайте новую таблицу данных (*Файл – >Создать*). Откройте модуль факторного анализа (*Анализ –> Многомерный разведочный анализ –> Факторный анализ*). (Переменные все, максимальное число факторов = 2).

6. Постройте график каменистой осыпи и сделайте соответствующие выводы о конечном количестве факторов.

7. Постройте График нагрузок. На получившемся графике наглядно можно увидеть взаимосвязь исследуемых параметров: сильно зависящие друг от друга параметры сконцентрированы в одной группе. Исследуйте каждую группу таких параметров и выведите главный – наиболее показательный. Удалите остальные переменные группы из дальнейшего

рассмотрения. При удалении переменных стоит ориентироваться на их пояснение в журнале производительности: в нем описывается значение и важность каждой. Повторяйте данный пункт до тех пор, пока не останется необходимого количества переменных.

8. Если на графике из-за наложения сложно разобрать названия переменных, то стоит обратиться к таблице факторных нагрузок из того же окна факторного анализа. В ней в числах выражена зависимость переменных от факторов. Группируя переменные по наиболее схожим показателям, мы можем понять, какие из них образуют группу на графике.

СОДЕРЖАНИЕ ОТЧЕТА

1. Описание принципа выбора счетчиков для наблюдения. Краткие описания каждого из них.
2. Описание условий, при которых снимались показания (запущенные действия и программы).
3. Исходная таблица данных в Excel.
4. Оценка дисперсии исследуемых параметров. Описание первого этапа сокращения данных.
5. Корреляционная матрица. Смысловое объяснение выявленных по матрице сильных зависимостей.
6. Результаты факторного анализа
   6.1 График каменистой осыпи.
   6.2 Описание каждого этапа сокращения данных: графики, таблицы собственных значений факторов, удаляемые переменные.
7. Выводы.

ПРИМЕР ВЫПОЛНЕНИЯ РАБОТЫ

Цель работы: выделение наиболее показательных системных счётчиков, которые косвенно могут давать нам информацию об остальных параметрах системы.

Выполнение:

Для выполнения практической работы №1 мы используем программное обеспечение Statsoft® STATISTICA, MS Excel. В качестве датасета будет использован датасет Absenteeism at work, взятый с сайта https://archive.ics.uci.edu/.

Датасет содержит следующие признаки:

1. Individual identification (ID).
2. Reason for absence (ICD).
3. Month of absence.
4. Day of the week (Monday (2), Tuesday (3), Wednesday (4), Thursday (5), Friday (6)).

5. Seasons.
6. Transportation expense.
7. Distance from Residence to Work (kilometers).
8. Service time.
9. Age.
10. Work load Average/day.
11. Hit target.
12. Disciplinary failure (yes=1; no=0).
13. Education (high school (1), graduate (2), postgraduate (3), master and doctor (4)).
14. Son (number of children).
15. Social drinker (yes=1; no=0).
16. Social smoker (yes=1; no=0).
17. Pet (number of pet).
18. Weight.
19. Height.
20. Body mass index.
21. Absenteeism time in hours (target).

Все признаки датасета являются числовыми, поэтому не требует преобразований. Рассчитав дисперсию в MS Excel мы не получили нулевых значений, поэтому будем рассматривать все признаки.

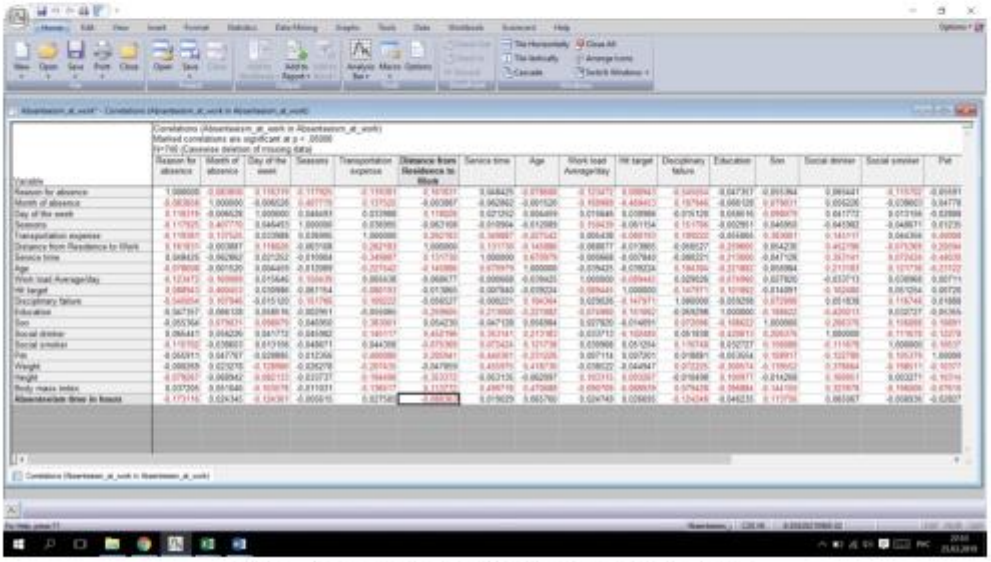


Рис. 1.1 Скриншот исходных данных

Рис. 1.2. Корреляционная матрица

Построим корреляционную матрицу (рис.1.2.) и выделим наиболее значимые коэффициенты корреляции, проанализируем их значения. По матрице корреляции можно увидеть, что наиболее коррелирующие параметры:

- Service time и Age (Стаж и Возраст).
- Вес и Индекс Массы тела

Далее согласно условию построим График каменистой осыпи (рис. 1.3).

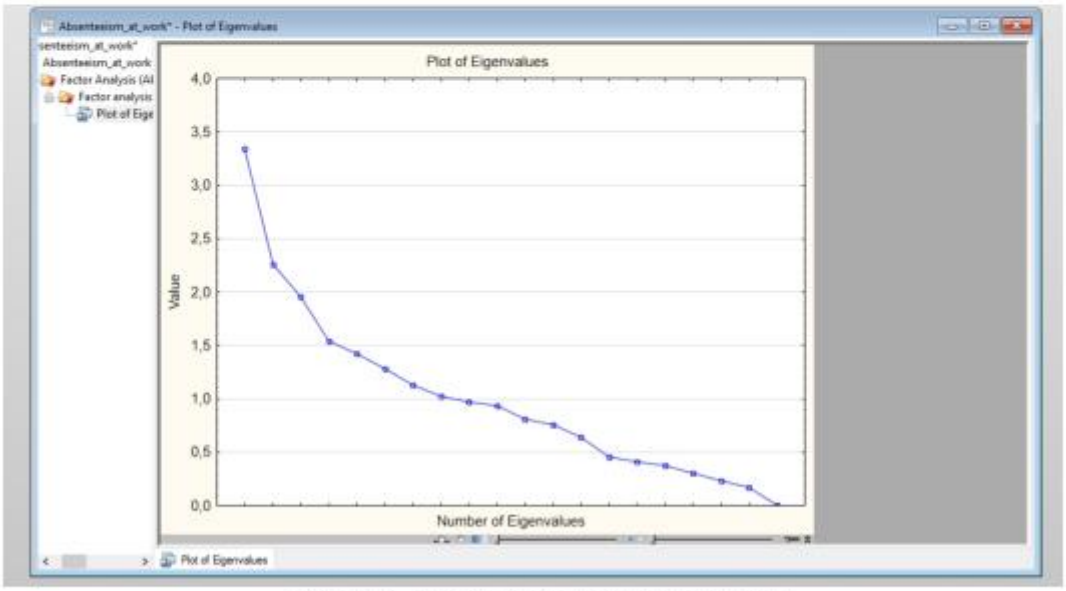


Рис. 1.3. График каменистой осыпи

Из полученного графика можно увидеть, что «перегиб» происходит на значении 4. Таким образом, можно выделить 4 фактора. Значит можно выделить 4 группы признаков.

Построим график нагрузок (рис. 1.4):

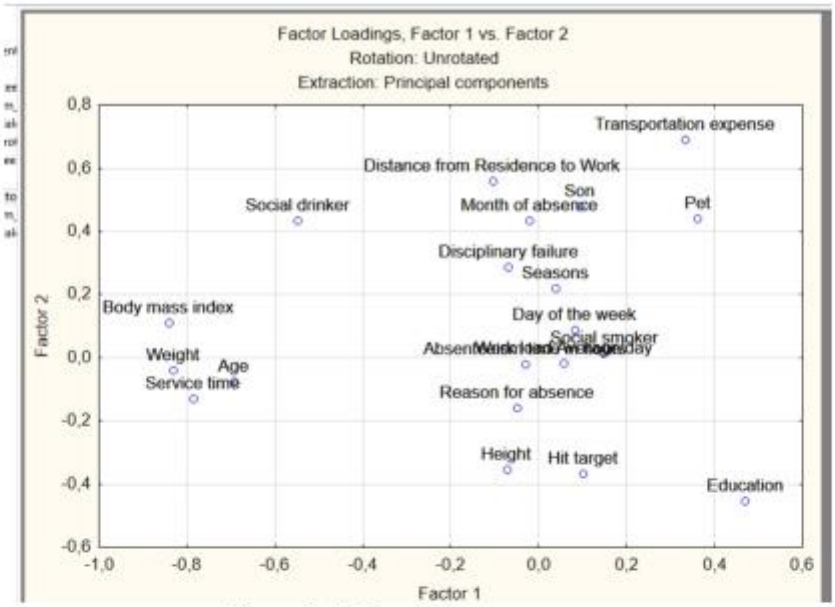


Рис. 1.4. График нагрузок

На графике отображены все параметры датасета. Мы видим, что они сгруппированы в разных областях. Из-за большого количества признаков сложно разобрать наименования параметров, а также непонятно, к какой области отнести тот или иной параметр, поэтому отобразим график нагрузок в формате 3D (рис. 1.5) и попробуем разметить группы признаков.

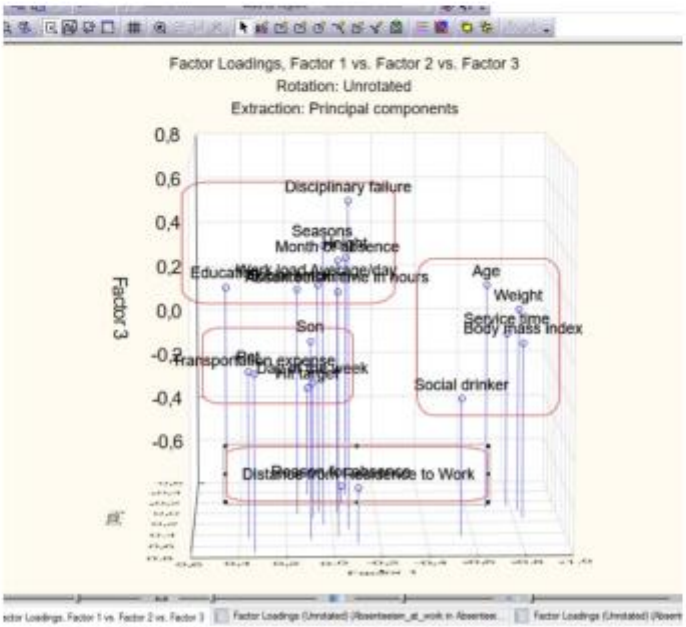


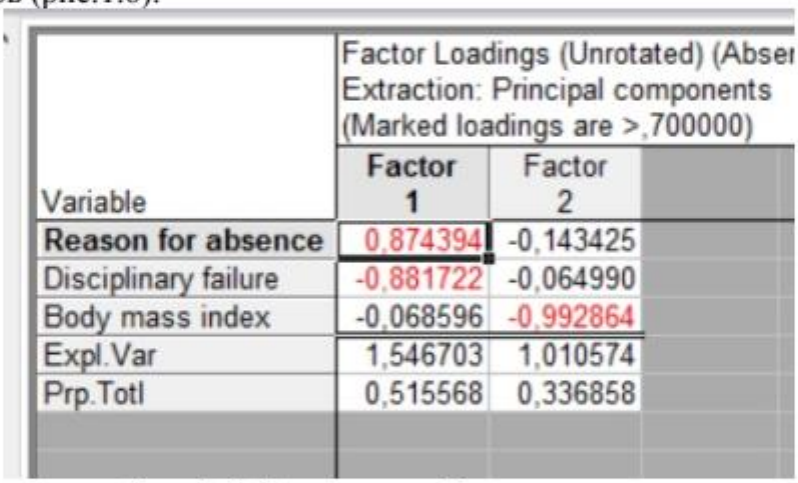
Рис. 1.5. График нагрузок в формате 3D

Таким образом, мы получили следующие группы признаков:

- Reasons for absence, day of the week, distance from residence to work;
- Month of absence, Seasons, Work load Average/day, Disciplinary failure, Son, Social smoker, Height, Absenteeism time in hours;
- Service time, Age, Social drinker, weight, body mass index;
- Transportation expense, Education, Pet.

Они формируются на основе того, что эти признаки взаимосвязаны, например, третья группа содержит индекс массы тела и вес - индекс массы тела вычисляется на основе веса, стаж работы зависит от возраста и к этим параметрам еще добавляется употребление алкоголя.

Попробуем уменьшить количество факторов, исключая те факторы, которые меньше всего влияют на целевой признак. Таким образом, получаем следующий результат, целевой признак очень сильно зависит от веса работников (рис.1.6).



Variable	Factor Loadings (Unrotated) (Abser Extraction: Principal components (Marked loadings are >,700000) Factor 1	Factor 2
Reason for absence	0,874394	-0,143425
Disciplinary failure	-0,881722	-0,064990
Body mass index	-0,068596	-0,992864
Expl.Var	1,546703	1,010574
Prp.Totl	0,515568	0,336858

Рис. 1.6. Скриншот таблицы нагрузок

Вывод:

Таким образом, проведя факторный анализ, мы выявили системные показатели, которые дают нам представление о системе признаков с помощью корреляционных матриц.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для решения каких задач применяется факторный анализ?
2. Что показывает корреляционная матрица?
3. Что такое вращение факторной структуры?
4. Какие методы вращения вы знаете?

ПРАКТИЧЕСКАЯ РАБОТА №2. ОТБОР ПРИЗНАКОВ

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Практически в любой задаче моделирования возникает вопрос: какую модель зависимости применить, т.е. какие признаки использовать, а какие нет? Проблема отбора признаков (features selection) возникает из-за того, что на этапах постановки задачи и формирования данных ещё не ясно, какие признаки действительно важны, а какие не несут полезной информации или дублируют друг друга. Стремление учесть как можно больше потенциально полезной информации приводит к появлению избыточных (шумовых) признаков. По мере увеличения числа используемых признаков (сложности модели) средняя ошибка на обучающей выборке, как правило, монотонно убывает. Однако средняя ошибка на независимых контрольных данных сначала уменьшается, а затем возрастает. Это явление называют переобучением. В чрезмерно сложных моделях избыточные степени свободы ¾расходуются не столько на восстановление искомой зависимости, сколько на аппроксимацию ошибок измерений и погрешностей самой модели. Отбор признаков позволяет находить модель оптимальной сложности, при которой переобучение минимально.

Для проверки качества отбора признаков используется критерий – функционал $Q\text{int}(\mu, X\ell)$, характеризующий качество метода μ по обучающей выборке $X\ell$, например, ошибка обучения (training error). Чем меньше значение критерия $Q(\mu)$, тем выше качество метода μ. Критерий должен быть внешним, т.е. проверять качество метода μ по тем данным, которые не использовались в процессе обучения. Наиболее известные типы внешних критериев:

- Критерий средней ошибки на контрольных данных.
- Критерий скользящего контроля: берут несколько различных разбиений исходной выборки на обучение и контроль, и среднюю ошибку на контроле усредняют по разбиениям. Обратите внимание на то, что во всех критериях, использующих случайные разбиения, обучающие и контрольные подвыборки должны обладать теми же статистическими характеристиками, что и полная выборка.
- Критерии непротиворечивости: если модель алгоритмов A и метод обучения μ подобраны правильно, то настройка параметров модели по различным представительным подвыборкам должна приводить к одинаковым или почти одинаковым алгоритмам.
- Критерии регуляризации – наложить ограничения на вектор параметров алгоритма, либо ввести штраф за выход вектора параметров из некоторой допустимой области (например, чтобы норма вектора параметров $\| a \|$ в алгоритме $a = \mu(X\ell)$ не становилась слишком большой).

Как правило, используется совокупность критериев. Практическая рекомендация – отобрать некоторое количество лучших методов по

критерию скользящего контроля; а из них выбрать тот, для которого критерий регуляризации (либо критерий непротиворечивости) принимает наименьшее значение.

Задача отбора информативных признаков состоит в следующем. Будем считать, что объекты описываются набором признаков $F = \{f1,..., fn\}$. Вектор $f1(x), \ldots, fn(x) \in D1 \times \ldots \times Dn$, где Dj –множество допустимых значений признака fj, называется признаковым описанием объекта x. Пусть $G \subseteq F$ произвольное подмножество признаков. Будем обозначать через μG метод обучения, который строит алгоритмы, используя только признаки из подмножества G . Будем предполагать, что метод μG выбирает алгоритм из модели алгоритмов A(G), использующей только признаки из G . Число используемых признаков $|G|$ будем называть сложностью модели A(G).

Для отбора информативных признаков используются различные методы:

- Полный перебор.

- Последовательное добавление признаков – простая стратегия жадного наискорейшего спуска: алгоритм добавляет к набору G по одному признаку, каждый раз выбирая тот признак, который приводит к наибольшему уменьшению внешнего критерия. Возможно также последовательное удаление признаков из полного набора, а также комбинация – алгоритм последовательного добавления–удаления.

- Построение и обход дерева возможных наборов признаков. Вершины дерева соответствуют наборам признаков. Корневая вершина соответствует пустому набору. Каждый дочерний набор образуется путём присоединения некоторого признака к родительскому набору. Чтобы избежать появления в дереве одинаковых наборов, отличающихся только порядком признаков, к дочерним наборам присоединяются только те признаки, номера которых превышают максимальный номер признака в родительском наборе. Как известно, существуют две стратегии полного обхода дерева: поиск в глубину (depth-first search, DFS) и поиск в ширину (breadth-first search, BFS). Обе позволяют вводить различные эвристики для сокращения перебора.

- Генетический алгоритм. Первое поколение наборов генерируется случайным образом. К этим наборам применяются операции скрещивания и мутации для порождения большого числа новых наборов. Затем производится .селекция: во второе поколение отбираются только B наборов, лучших по заданному внешнему критерию Q. Ко второму поколению также применяются операции скрещивания, мутации и селекции, и порождается третье поколение. Эволюционный процесс переходит от поколения к поколению до тех пор, пока не наступит стагнация, т.е. качество лучшего набора в поколении перестанет улучшаться.

- Случайный поиск с адаптацией. Если упростить генетический алгоритм, отказавшись от скрещивания, то получится алгоритм случайного поиска (stochastic search).

- Кластеризация признаков. Методы кластеризации в общем случае позволяют разбить выборку объектов на кластеры, состоящие из схожих объектов, и выделить в каждой группе по одному наиболее типичному представителю. То же самое можно проделать и с признаками, если определить функцию расстояния между признаками, например, через коэффициент корреляции или метрику Хемминга.

- Методы математического программирования. Используются для отбора признаков, главным образом, в линейных моделях регрессии и классификации. По существу, здесь также реализуется перебор признаков, но перебор внутри стандартных процедур математического программирования при поиске активных ограничений.

Более подробно со спецификой применения методов можно ознакомиться [3], в разделе «Критерии выбора моделей и методы отбора признаков».

ЦЕЛЬ РАБОТЫ

Реализовать алгоритм по отбору признаков на выбранном наборе данных.

ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ И ПОРЯДОК ВЫПОЛНЕНИЯ

1. Выбрать предметную область и набор данных, согласовать выбор с преподавателем. Можно использовать репозиторий ресурса http://archive.ics.uci.edu/ml/.
2. Выбрать алгоритм отбора признаков, согласовать выбор с преподавателем.
3. Реализовать алгоритм по отбору признаков на выбранном наборе данных. Выбор способа реализации алгоритма предоставляется студенту.
4. Проверить качество реализованного отбора признаков с помощью одного из критериев.

ПРИМЕР ВЫПОЛНЕНИЯ РАБОТЫ

Для выполнения лабораторной работы был выбран датасет, содержащий показатели исследуемых пациентов, у которых был диагностирован рак молочной железы. Датасет взят из архива UCI: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Coimbra#.

Описание отобранного датасета:

Существует 10 предикторов, все количественные, и бинарная зависимая переменная, указывающая на наличие или отсутствие рака молочной железы. Предикторами являются антропометрические данные и параметры, которые могут быть собраны при анализе крови.

Атрибуты:
1. Age (years) – возраст.
2. BMI (kg/m2) – индекс массы тела.
3. Glucose (mg/dL) – уровень сахара в крови.
4. Insulin (µU/mL) – содержание инсулина в организме.
5. HOMA – индекс инсулинорезистентности.
6. Leptin (ng/mL) – содержание лептина в организме.
7. Adiponectin (µg/mL) – гормон адипонектин.
8. Resistin (ng/mL) – гормон резистин.
9. MCP-1(pg/dL) – содержание компонента MCP-1.

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1	Classification
0	48	23.500000	70	2.707	0.467409	8.8071	9.702400	7.99585	417.114	1
1	83	20.690495	92	3.115	0.706897	8.8438	5.429285	4.06405	468.786	1
2	82	23.124670	91	4.498	1.009651	17.9393	22.432040	9.27715	554.697	1
3	68	21.367521	77	3.226	0.612725	9.8827	7.169560	12.76600	928.220	1
4	86	21.111111	92	3.549	0.805386	6.6994	4.819240	10.57635	773.920	1

Рис. 2.1. Пример данных из датасета

Импортируем необходимые модули и библиотеки:

In [1]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
import pandas as pd
import seaborn as sns
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, LassoCV, Ridge, RidgeCV
from sklearn.tree import DecisionTreeClassifier, export_graphviz

%matplotlib inline
data = pd.read_csv('dataR2.csv')
data.head()
```

Для работы с данными осуществим необходимые преобразования:

In [2]:
```
categorical_columns = [c for c in data.columns if data[c].dtype.name == 'object']
numerical_columns   = [c for c in data.columns if data[c].dtype.name != 'object']
print(categorical_columns)
print(numerical_columns)
```

```
[]
['Age', 'BMI', 'Glucose', 'Insulin', 'HOMA', 'Leptin', 'Adiponectin', 'Resistin', 'MCP.1', 'Classification']
```

In [3]:
```
data_numerical = data[numerical_columns]
data_numerical = (data_numerical - data_numerical.mean()) / data_numerical.std()
data_numerical.describe()
```

Out[3]:

	Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1	C
count	1.160000e+02	1.160000e+02	1.160000e+02	1.160000e+02	1.160000e+02	1.160000e+02	1.160000e+02	1.160000e+02	1.160000e+02	1.
mean	1.397350e-16	-2.704135e-15	2.660707e-16	5.665966e-16	7.465293e-17	1.110223e-16	-2.268300e-16	1.081510e-16	-8.384098e-16	9.
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.
min	-2.066791e+00	-1.835032e+00	-1.677817e+00	-7.529064e-01	-6.116289e-01	-1.162682e+00	-1.245715e+00	-9.294081e-01	-1.413085e+00	-1
25%	-7.634769e-01	-9.180839e-01	-5.346511e-01	-5.614786e-01	-4.879188e-01	-7.455135e-01	-6.877622e-01	-6.330746e-01	-7.651317e-01	-1
50%	-8.078837e-02	1.599667e-02	-2.571837e-01	-4.060072e-01	-3.607999e-01	-3.307086e-01	-2.671475e-01	-3.146104e-01	-1.830650e-01	8.
75%	8.501505e-01	7.289308e-01	1.867643e-01	1.169240e-01	4.470017e-02	5.610726e-01	2.389324e-01	2.444781e-01	4.782652e-01	8.
max	1.957277e+00	2.190508e+00	4.581849e+00	4.812180e+00	6.138135e+00	3.318769e+00	4.070983e+00	5.437492e+00	3.364413e+00	8.

Отбор признаков будем осуществлять на основе их важности. Первым методом выберем линейную регрессию. Используем функцию LinearRegression. Для оценки точности алгоритма линейной регрессии будем использовать RMSE.

```
In [8]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 11)

In [9]: lin = LinearRegression(normalize=True)
        lin.fit(X_train, y_train)

Out[9]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=True)

In [10]: y_lin_pred = lin.predict(X_test)

In [11]: def rmse(y, p):
             return np.sqrt(mean_squared_error(y, p))
         def beatiful_coef(coefs, feature_names=data.columns):
             return pd.DataFrame(coefs, index=feature_names,
                                 columns=['coef']).sort_values('coef',
                                                               ascending=False)

In [12]: rmse(y_test, y_lin_pred)

Out[12]: 0.9861422652300279
```

Полученный коэффициент точности равен 0,98614.
В результате проделанной работе получили следующий результат:

```
In [15]: beatiful_coef(ridge.coef_,feature_names=X_train.columns)
```

Out[15]:

	coef
Glucose	0.136434
Resistin	0.083716
Insulin	0.063370
HOMA	0.033702
MCP.1	0.030060
Adiponectin	0.004415
Age	-0.025034
Leptin	-0.036648
BMI	-0.086257

Самыми показательными параметрами получились содержание сахара в крови и гормон резистин.В качестве второго метода для отбора признаков выберем метод случайного леса.

Ансамблевые алгоритмы на основе деревьев решений, такие как случайный лес (random forest), позволяют оценить важность признаков.

```
In [24]: from sklearn.ensemble import RandomForestRegressor
         forest = RandomForestRegressor(n_estimators=100, random_state=17)
         forest.fit(X_train, y_train)
         forest_test_pred = forest.predict(X_test)

         C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boostin
         th_tests is an internal NumPy module and should not be imported. It will b
           from numpy.core.umath_tests import inner1d

In [25]: rmse(y_test, forest_test_pred)

Out[25]: 0.6212527085531415

In [26]: beatiful_coef(forest.feature_importances_,feature_names=X_train.columns)
```

Out[26]:

	coef
Resistin	0.187068
Glucose	0.163129
Age	0.162173
BMI	0.127742
Adiponectin	0.096024
Leptin	0.087352
HOMA	0.073114
MCP.1	0.060333
Insulin	0.043066

В результате выявили два важных признака: резистин и содержание сахара в крови. Получили результат RMSE равный 0.6213, что намного меньше в сравнении с оценкой алгоритма линейной регрессии. Значит, метод случайного леса является более точным для отбора признаков.

Выводы

Сравнив результаты двух методов отбора признаков (линейная регрессия и случайный лес), мы получили схожие результаты. Оба метода указывают на важность признаков Glucose и Resistin, но точность метода Random Forest больше, чем метода Линейной регрессии. В данном случае, учитывая специфику выборки, имеет смысл брать выявленные признаки в рассмотрение. Подводя итог, было отобрано 2 признака из девяти рассматриваемых. Сокращение числа признаков увеличит скорость обработки данных и качество результатов, так как малоинформативные признаки будут отброшены.

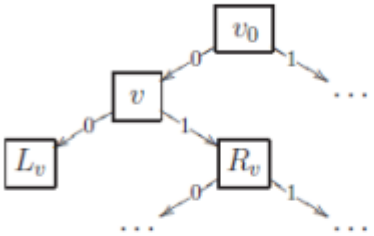
КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое нормализация данных?
2. Перечислите основных преимущества отбора признаков?
3. В чем основная идея отбора признаков с помощью генетического алгоритма?
4. В чём отличия внутренних и внешних критериев?

ПРАКТИЧЕСКАЯ РАБОТА №3. ДЕРЕВЬЯ РЕШЕНИЙ

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Бинарное решающее дерево – это алгоритм классификации, задающийся бинарным деревом, в котором каждой внутренней вершине $v \in V$ приписан предикат $\beta_v : X \to \{0, 1\}$, каждой терминальной вершине $v \in V$ приписано имя класса $c_v \in Y$. При классификации объекта $x \in X$ он проходит по дереву путь от корня до некоторого листа, в соответствии с Алгоритмом 1.



Алгоритм 1. Классификация объекта $x \in X$ бинарным решающим деревом

```
1: v := v0;
2: пока вершина v внутренняя
3: если βv(x) = 1 то
4: v := Rv; (переход вправо)
5: иначе
6: v := Lv; (переход влево)
7: вернуть cv.
```

Объект x доходит до вершины v тогда и только тогда, когда выполняется конъюнкция $K_v(x)$, составленная из всех предикатов, приписанных внутренним вершинам дерева на пути от корня v_0 до вершины v. Естественное требование максимизации информативности конъюнкций $K_v(x)$ означает, что каждая из них должна выделять как можно больше обучающих объектов, допуская при этом как можно меньше ошибок. Задача построения дерева минимальной сложности, правильно классифицирующего заданную выборку, в общем случае является NP-полной задачей. На практике применяют различные эвристики.

В данной практической работе рассматривается алгоритм 2 построения решающего дерева ID3 (Induction of Decision Tree). Идея алгоритма заключается в последовательном дроблении выборки на две части до тех пор, пока в каждой части не окажутся объекты только одного класса. Алгоритм записывается в виде рекурсивной процедуры LearnID3, которая строит дерево по заданной подвыборке U. Для построения полного дерева она применяется ко всей выборке и возвращает указатель на корень построенного дерева:

v0 := LearnID3 (Xℓ).

Алгоритм 2. Рекурсивный алгоритм синтеза бинарного решающего дерева ID3

```
Вход: U – обучающая выборка; B – множество элементарных предикатов;
Выход: возвращает корневую вершину дерева, построенного по выборке U;
1:  ПРОЦЕДУРА LearnID3 (U);
2:  если все объекты из U лежат в одном классе c ∈ Y то
3:  создать новый лист v;
4:  cv := c;
5:  вернуть (v);
6:  найти предикат с максимальной информативностью:
      β := arg max I(β, U); β∈B
7:  разбить выборку на две части U = U0∪U1 по предикату β:
    U0 := {x ∈ U : β(x) = 0};
    U1 := {x ∈ U : β(x) = 1};
8:  если U0 = ∅ или U1 = ∅ то
9:  создать новый лист v;
10: cv := класс, в котором находится большинство объектов из U;
11: иначе
12: создать новую внутреннюю вершину v;
13: βv := β;
14: Lv := LearnID3 (U0); (построить левое поддерево)
15: Rv := LearnID3 (U1); (построить правое поддерево)
16: вернуть (v);
```

Проблемы, возникающие при реализации алгоритма:

П1. Как определить информативность предиката? – 2 варианта.

Точный критерий Фишера основан на вычислении вероятности реализации пары событий: информативность предиката φ(x), разделяющего события правильного обнаружения p_c нужных объектов и правильного необнаружения N_c ненужных объектов, относительно класса c ∈ Y по выборке X^ℓ

$$I_c(\varphi, X^\ell) = -\ln \frac{C_{P_c}^{p_c(\varphi)} C_{N_c}^{n_c(\varphi)}}{C_{P_c+N_c}^{p_c(\varphi)+n_c(\varphi)}}.$$

в случае произвольного числа классов Y = {1, . . . ,M}

$$I(\varphi, X^l) = -\ln \frac{C_{P1}^{p1} \ldots C_{PM}^{pM}}{C_l^p} \qquad (*)$$

где P_c - число объектов класса c в выборке X^ℓ, из них p_c объектов выделяются предикатом φ, p = p1+…+ pM.

Энтропийный = информационный критерий основан на сравнении энтропии (математического ожидания количества информации) о выборке до и после применения предиката φ. Если в выборке есть P объектов класса c и N остальных, то ее энтропия равна

$$\overset{)}{H}(P,N) = -\frac{P}{P+N}\log_2\frac{P}{P+N} - \frac{N}{P+N}\log_2\frac{N}{P+N}.$$

Если предикат φ выделил p объектов из P, принадлежащих классу c, и n объектов из N, не принадлежащих классу c, то энтропия выборки после получения этой информации стала равной

$$\overset{)}{H}_\varphi(P,N,p,n) = \frac{p+n}{P+N}\overset{)}{H}(p,n) + \frac{P+N-p-n}{P+N}\overset{)}{H}(P-p,N-n).$$

Тогда энтропийный = информационный критерий для двух классов вводится как

$$IGain(\varphi, X^\ell) = \overset{)}{H}(P,N) - \overset{)}{H}(P,p,N,n),$$

а для большого числа классов – как

$$I(\varphi, X^\ell) = \sum_{c\in Y} h\left(\frac{P_c}{\ell}\right) - \frac{p}{\ell}\sum_{c\in Y} h\left(\frac{p_c}{p}\right) - \frac{\ell-p}{\ell}\sum_{c\in Y} h\left(\frac{P_c-p_c}{\ell-p}\right),$$

где введена функция $h(z) \equiv -z\log_2 z$.

(**)

П2. На шаге 6 Алгоритма 2 выбирается предикат β из заданного семейства B, задающий максимально информативное ветвление дерева – разбиение выборки на две части $U = U_0 \cup U_1$.

На практике применяются различные критерии ветвления.

1. Критерий, ориентированный на отделение заданного класса $c \in Y$.
$I(\beta, U) = \max I_c(\beta, U)$, $c \in Y$.
2. Критерии, ориентированные на отделение сразу нескольких классов – (*) и (**).
3. D-критерий _ число пар объектов из разных классов, на которых предикат β принимает разные значения. В случае двух классов он имеет вид
$I(\beta, U) = p(\beta)(N - n(\beta)) + n(\beta)(P - p(\beta))$.

Другие варианты критериев можно найти в [2].

П3. Как выбрать условия деления выборки в каждом узле?

На практике в качестве элементарных предикатов чаще всего берут простые пороговые условия вида $\beta(x) = [f_j(x) \gtrless d_j]$. Конъюнкции, составленные из таких термов, хорошо интерпретируются и допускают запись на естественном языке. Однако никто не запрещает использовать в вершинах дерева любые разделяющие правила: шары, гиперплоскости, и, вообще говоря, произвольные бинарные классификаторы. Когда мощность |B| не велика, эта задача решается полным перебором, дальше нужны эвристики.

П4. Как удалить поддеревья, имеющие недостаточную статистическую надёжность? = проблема редукции решающих деревьев.

Наиболее распространенные эвристики:

Предредукция (pre-pruning) или критерий раннего останова досрочно прекращает дальнейшее ветвление в вершине дерева, если информативность $I(\beta, U)$ для всех предикатов $\beta \in B$ ниже заданного порогового значения I_0. Для этого на шаге 8 условие ($U0 = \varnothing$ или $U1 = \varnothing$) заменяется условием $I(\beta, U) \le I_0$. Порог I_0 является управляющим параметром метода.

Постредукция (post-pruning) просматривает все внутренние вершины дерева и заменяет отдельные вершины либо одной из дочерних вершин (при этом вторая дочерняя удаляется), либо терминальной вершиной. Процесс замен продолжается до тех пор, пока в дереве остаются вершины, удовлетворяющие критерию замены. Критерием замены является сокращение числа ошибок на контрольной выборке, отобранной заранее, и не участвовавшей в обучении дерева. Стандартная рекомендация - оставлять в контроле около 30% объектов.

Для реализации постредукции контрольная выборка Xk пропускается через построенное дерево. При этом в каждой внутренней вершине v запоминается подмножество $S_v \subseteq X_k$ попавших в неё контрольных объектов. Если $Sv = \varnothing$, то вершина v считается ненадёжной и заменяется терминальной по мажоритарному правилу: в качестве c_v берётся тот класс, объектов которого больше всего в обучающей подвыборке U, пришедшей в вершину v. Затем для каждой внутренней вершины v вычисляется число ошибок, полученных при классификации выборки Sv следующими способами:

1) $r(v)$ - классификация поддеревом, растущим из вершины v;
2) $r_L(v)$ - классификация поддеревом левой дочерней вершины L_v;
3) $r_R(v)$ - классификация поддеревом правой дочерней вершины R_v;
4) $r_c(v)$ - отнесение всех объектов выборки S_v к классу $c \in Y$.

Эти величины сравниваются, и, в зависимости от того, какая из них оказалась минимальной, принимается, соответственно, одно из четырёх решений:

1) сохранить поддерево вершины v;
2) заменить поддерево вершины v поддеревом левой дочерней вершины Lv;
3) заменить поддерево вершины v поддеревом правой дочерней вершины Rv;
4) заменить поддерево v терминальной вершиной класса $c = \arg\min_{c \in Y} r_c(v)$.

Предпросмотр (look ahead) заключается в том, чтобы на шаге 6, вместо вычисления информативности для каждого $\beta \in B$ построить поддерево небольшой глубины h. Во внутреннюю вершину v помещается тот предикат β, при котором поддерево допускает наименьшее число ошибок. Этот алгоритм работает заметно дольше, но строит более надёжные и простые

деревья. Более подробно со спецификой применения методов можно ознакомиться в [8].

ЦЕЛЬ РАБОТЫ

Реализовать и сравнить два разных алгоритма деревьев решений.

ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ И ПОРЯДОК ВЫПОЛНЕНИЯ

1. Выбрать статистический ряд, подходящий для построения дерева решений, выделить на нем обучающую и контрольную выборку, согласовать с преподавателем.
2. Реализовать алгоритм 2. В ходе реализации алгоритма выбрать способы решения проблем П1-П4, причем хотя бы одну из проблем решить двумя способами.

ПРИМЕР ВЫПОЛНЕНИЯ РАБОТЫ

Цель работы: на выбранном наборе данных реализовать алгоритм случайного дерево и случайного леса, классифицировать важность влияния признаков на целевой признак. Сделать соответствующие выводы.

Практическая работа выполнена на наборе данных Parkinson Dataset with replicated acoustic features Data Set.

Ссылка:
http://archive.ics.uci.edu/ml/datasets/Parkinson+Dataset+with+replicated+acoustic+features+

Аннотация: набор данных содержит акустические характеристики, извлеченные из 3-х голосовых записей, на которые записана непрерывная речь каждого из 80 объектов исследования (40 из которых имеют болезнь Паркинсона).

Датасет содержит следующие признаки:

1. **ID**: уникальный идентификатор объекта исследования.
2. **Recording**: порядковый номер голосовой записи.
3. **Status**: 0=здоров; 1= болезнь Паркинсона
4. **Gender**: 0=муж; 1=жен
5. **Pitch local perturbation measures** (фазовое дрожание цифрового сигнала данных):

– **Relative jitter** (Jitter_rel): средняя абсолютная разница между последовательными периодами, деленная на средний период

– **Absolute jitter** (Jitter_abs): изменение основной частоты между циклами (средняя абсолютная разница между последовательными периодами).

– **Relative average perturbation** (Jitter_RAP): относительное среднее возмущение (средняя абсолютная разница между периодом и средним значением для него и его двух соседей, деленная на средний период).

– **Pitch perturbation quotient** (Jitter_PPQ): коэффициент возмущения периода (средняя абсолютная разница между периодом и средним значением, деленная на средний период).

6. **Amplitude perturbation measures**:

– **local shimmer** (Shim_loc): средняя абсолютная разница между амплитудами двух последовательных периодов, деленных на среднюю амплитуду. 3,81% - предел обнаружения паталогий.

– **shimmer in dB** (Shim_dB): средняя абсолютная разница от ln разности между двумя последовательными периодами. Предел для обнаружения патологий составляет 0,350 дБ.

– **3-point amplitude perturbation quotient** (Shim_APQ3): это трехточечный коэффициент дрожания амплитуды, средняя абсолютная разница между амплитудой периода и средней амплитудой его соседей, деленная на среднюю амплитуду.

– **5-point amplitude perturbation quotient** (Shim_APQ5): это коэффициент дрожания амплитуды из пяти точек, средняя абсолютная разница между амплитудой периода и средней амплитуды его и его четырех ближайших соседей, деленная на среднюю амплитуду.

– **11-point amplitude perturbation quotient** (Shim_APQ11): это коэффициент дрожания амплитуды, равный 11 точкам, средняя абсолютная разница между амплитудой периода и средней амплитуды его и его десяти ближайших соседей, деленная на среднюю амплитуду. Параметр APQ и дает 3,070% в качестве порога для патологии.

7. **Harmonic-to-noise ratio measures**: (отношение гармоник к шуму)
   – in the frequency band 0-500 Hz (HNR05).
   – in 0-1500 Hz (HNR15).
   – in 0-2500 Hz (HNR25).
   – in 0-3500 Hz (HNR35).
   – in 0-3800 Hz (HNR38).
8. **Mel frequency cepstral coefficient-based spectral measures** (еденица высоты тона) of order 0 to 12 (MFCC0 – MFCC12) and their **derivatives** (производные) (Delta0 – Delta12).
9. **Recurrence period density entropy** (RPDE): энтропия плотности периода повторения.
10. **Detrended fluctuation analysis** (DFA): метод Пенга или анализ отклонения колебаний.

11. **Pitch period entropy** (PPE): энтропия основного периода.
12. **Glottal-to-noise excitation ratio** (GNE): соотношение возбуждения гортань-шум.

Импортируем библиотеки, необходимые для выполнения практической работы:

```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.model_selection import KFold
        from sklearn.model_selection import cross_val_score
```

Укажем путь к датасету и импортируем его:

```
In [2]: df = pd.read_csv('C:/Users/Po/Downloads/ReplicatedAcousticFeatures-Parkinson
```

Модифицируем датасет:

```
In [3]: del df['ID']
        del df['Recording']
        df.head()
```

Out[3]:

	Status	Gender	Jitter_rel	Jitter_abs	Jitter_RAP	Jitter_PPQ	Shim_loc	Shim_dB	Shim_AF
0	0	1	0.25546	0.000015	0.001467	0.001673	0.030256	0.26313	0.017
1	0	1	0.36964	0.000022	0.001932	0.002245	0.023146	0.20217	0.013
2	0	1	0.23514	0.000013	0.001353	0.001546	0.019338	0.16710	0.011
3	0	0	0.29320	0.000017	0.001105	0.001444	0.024716	0.20892	0.014
4	0	0	0.23075	0.000015	0.001073	0.001404	0.013119	0.11607	0.006

5 rows × 46 columns

1. Реализуем алгоритм «Случайное дерево».

```
In [6]: clf_tree = DecisionTreeClassifier(random_state=17)

In [7]: df1 = df['Status'] # целевой столбец
        del df['Status']

In [8]: clf_tree.fit(df, df1)

Out[8]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                    max_features=None, max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, presort=False, random_state=17,
                    splitter='best')
```

In [9]:
```
feature_importances = pd.DataFrame(clf_tree.feature_importances_,
                                   index = df.columns,
                                   columns=['importance']).sort_values('importance',ascending=False)
```

In [10]:
```
feature_importances.head(10)
```

Out[10]:

	importance
Delta0	0.375076
Shim_loc	0.102298
MFCC4	0.076957
HNR38	0.055652
PPE	0.049365
MFCC3	0.034794
Delta5	0.027778
Delta8	0.026825
RPDE	0.026813
Delta11	0.026804

In [11]:
```
kf = KFold(shuffle = True, random_state = 17, n_splits = 4)
```

In [12]:
```
score = cross_val_score(estimator = clf_tree, X = df, y = df1, cv=kf, scoring ="accuracy")
```

In [13]:
```
print(score)
score.mean()
```

```
[0.7        0.7        0.73333333 0.65       ]
```

Out[13]:
```
0.6958333333333333
```

2. Реализуем алгоритм «Случайный лес».

In [15]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [16]:
```
clf = RandomForestClassifier(random_state=17, n_estimators = 100, n_jobs=-1)
```

In [18]:
```
clf.fit(df, df1)
```

Out[18]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
            oob_score=False, random_state=17, verbose=0, warm_start=False)
```

In [19]:
```
feature_importances = pd.DataFrame(clf.feature_importances_,
                                   index = df.columns,
                                   columns=['importance']).sort_values('importance',ascending=False)
```

In [20]:
```
feature_importances.head(10)
```

Out[20]:

	importance
HNR35	0.049012
MFCC4	0.046259
HNR38	0.045557
Delta11	0.044276
MFCC10	0.041315
Delta0	0.038235
Delta3	0.037973
Delta7	0.035280
MFCC11	0.034364
Delta1	0.033248

In [17]:
```
kf = KFold(shuffle = True, random_state = 17, n_splits = 4)
score = cross_val_score(estimator = clf, X = df, y = df1, cv=kf, scoring ="accuracy")
print(score)
print(score.mean())
```

```
[0.83333333 0.83333333 0.81666667 0.71666667]
0.8
```

Выводы

При применении первого алгоритма мы выявили признаки, которые влияют на систему, но не дают достаточной точности при вычислении. В исследовании с применением второго алгоритма непосредственное значение важности признаков упало, но точность стала достаточной для заключения о качестве работе алгоритма. Некоторые признаками, проявились при исследовании влияния дважды. Из значимых признаков были выделены следующие:

– отношение гармоник к шуму на высоких частотах 0-3500 Гц и 0-3800 Гц (HNR38 и HNR35);

Гармоника – это элементарная составляющая сложного гармонического колебания (сигнала). Отношение "сигнал/шум" – это отношение среднеквадратического значения величины входного сигнала к среднеквадратическому значению величины шума, выраженное в децибелах, данное значение позволяет определить долю шума в измеряемом сигнале по отношению к полезному сигналу.

– высота тона и производная этого значения (MFCC11 и Delta11).

Данный параметр измеряется в психофизических единицах высоты звука *Мел*. Он основан на статистической обработке большого числа данных о субъективном восприятии высоты звуковых тонов. С помощью формулы перевода можно преобразовать значение частоты звука (Гц) в значение высоты (мел). График данной зависимости имеет экспоненциальную форму, поэтому производная данной характеристики также выделяется значимым

параметром, влияющим на систему, ведь при экспонента мало изменяющаяся функция.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как деревья решений, так и глубокие нейронные сети являются нелинейными классификаторами, т. е. они разбивают пространство посредством сложной границы решений. Почему в таком случае модель дерева решений настолько интуитивно понятнее глубокой нейронной сети?
2. Чем отличаются алгоритмы деревья решения от других алгоритмов классификации?
3. Опишите достоинства и недостатки решающих деревьев?
4. Что такое решающее дерево?

ПРАКТИЧЕСКАЯ РАБОТА №4. НЕЙРОСЕТЕВОЕ РАСПОЗНАВАНИЕ ПЕЧАТНЫХ СИМВОЛОВ

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. Подготовка эталонных образов

Набор эталонных образов задается преподавателем. Примером такого набора является последовательность из десяти цифр от 0 до 9. В этом примере число образов M=10. В случае, когда каждый класс образов характеризуется лишь своим эталоном, имеем число классов, также равное M.

Каждый образ формируется в виде графического файла в битовом формате. Тип файла (расширение) определяется используемыми в среде MATLAB типами графических файлов. Рекомендуется использовать расширение ***tif***.

Для создания графических файлов образов удобно использовать среду "***Adobe Photoshop***". В этом случае при создании каждого файла необходимо проделать следующую последовательность операций:

1) создать новый файл, задав его параметры:

- имя : *XXXX*;
- ширина: N1 пикселей;
- высота: N2 пикселей;
- цветовой режим: битовый формат.

При этом значения N1, N2=8…20 задаются преподавателем.

2) используя инструменты типа «Кисть», «Ластик» и др. создать требуемый образ символа.

3) с помощью команды «Сохранить как» сохранить созданный образ в виде файла типа ***tif*** в заданной преподавателем папке.

На рис. 4.1 приведены примеры графических символов цифр при N1=10, N2=12 пикс.

Рис. 4.1. Примеры графических символов цифр

2. Создание и обучение НС в среде MATLAB

На данном этапе выполнение работы в среде MATLAB производится с помощью программы ***sr_newff*** , которая реализует следующие функции:

- формирование числовых массивов эталонных образов, используемых в качестве обучающих;

- подготовка данных, необходимых для создания нейронной сети (НС);

- создание НС, задание параметров обучения НС и обучение НС.

Эталонный образ каждого символа представлен в виде вектора-столбца [N,1], число элементов N которого равно числу признаков (иначе говоря, N – размерность пространства признаков). Такой вектор-столбец формируется из двумерного массива-изображения [N1,N2], который, в свою очередь, формируется при считывании графического файла образа с помощью команд:

imread (FILENAME) - процедура чтения графического файла;

X = reshape (A,[N,1]) - процедура преобразования двумерного массива A[N1,N2] в одномерный вектор-столбец X[N,1], где N=N1*N2.

Процедура умножения массива на 1 приводит к смене типа элементов массива с *logical* (для элементов битового формата) на *double*.

Для удовлетворительной работы НС недостаточно формирования лишь одного обучающего образа для каждого класса (типа символа) образов. Это связано с тем, что распознаваемые образы (на этапе работы НС в режиме распознавания) всегда отличаются от обучающих по ряду причин:

- различие шрифтов и стилей печатных символов;

- погрешности сканирования и неточности совмещения символа и окна сканирования;

- низкое качество печати, дефекты бумаги и т.д.

В силу указанных причин для надежного распознавания образов НС следует обучать на достаточно представительном множестве образов, входящих в один и тот же класс.

В программе ***sr_newff*** формирование дополнительных обучающих образов производится путем незначительного искажения эталонных образов, считываемых из графических файлов. Искажение образа-эталона каждого класса реализуется путем добавления к нему равномерного (по площади изображения) шума типа «Соль и перец», представляющего собой случайное искажение отдельных пикселей изображения. Степень искажения характеризуется числом p=[0;1], определяющим долю искаженных пикселей.

Такой подход при формировании образов позволяет, во-первых, быстро получать большое число обучающих образов, и, во-вторых, регулировать (путем изменения значения p) степень разброса множества образов в пределах одного класса.

Подготовка данных, необходимых для создания НС, включает в себя:

1) формирование двумерного массива обучающих образов XR[N,K], каждый столбец которого представляет собой набор N признаков одного образа, а число столбцов K равно числу обучающих образов;

2) формирование двумерного массива желаемых откликов YR[NY,K], где NY – число выходов НС (т.е., число нейронов выходного слоя); K – число

обучающих образов. Отклик YR[:,k] (в общем случае – вектор-столбец) соответствует k-му обучающему образу – вектору XR[:,k];
3) формирование двумерного массива R[N,2], определяющего минимальное R(n,1) и максимальное R(n,2) значение n-го признака, n=1,…,N.

Создание НС. В общем случае НС ***net*** создается с помощью команды:

net = nnnnn (P1,P2,…PL), где

- nnnnn – тип НС;
- P1,…,PL – параметры НС.

В настоящей работе используется НС типа многослойного персептрона newff, которая задается командой:

net = newff (R, [A1 A2 … AL], {F1 F2 … FL}, BTF, PF), где

R - массив минимальных и максимальных значений входных нейронов (признаков);
Ai - число нейронов i-го слоя, начиная с первого скрытого слоя, i=1,…,L;
Fi - функция активации нейронов i-го слоя, по умолчанию 'tansig';
BTF - функция обучения сети, по умолчанию 'trainlm';
PF - критерий остановки, по умолчанию 'mse' (минимум ско).

Дополнительные параметры, используемые при создании сети:
net.performFcn='msereg' - обучение НС производится с помощью метода регуляризации;
net.performParam.ratio=0.1 - значение параметра регуляризации;
net.trainParam.show=5 - число эпох, через которое производится вывод параметров процедуры обучения;
net.trainParam.epochs=500 - максимальное число эпох при обучении НС;
net.trainParam.goal=0.02 - значение целевой функции, по достижении которого процесс обучения НС прекращается.

Процесс обучения НС запускается командой:

net = train (net, XR, YR) .

Для решения задач распознавания печатных символов рекомендуется использовать трехслойную НС (один скрытый слой) с числом нейронов:

N=120 - во входном слое;
A1=20 - в скрытом (промежуточном) слое;
A2=1 - в выходном слое.

При использовании большего числа нейронов процедура обучения НС может занять слишком много времени. Для рекомендованных значений параметров НС (в том числе и дополнительных) и общем числе обучающих образов (для всех заданных классов) K=100…200 время обучения НС составляет 5…20 мин.

3. Распознавание печатных символов с помощью обученной НС

Работа НС, т.е. формирование отклика Y при входном воздействии в виде вектора-столбца X[N,1] производится командой:

Y = sim (net, X).

В случае, когда желаемый отклик принимает целочисленные значения, рекомендуется использовать округление до ближайшего целого, т.е.

Y = round (sim (net, X)).

Тестирование работы НС при распознавании печатных символов с различной степенью искажения производится с помощью программы ***sr_work*** , исходными данными для которой являются:

SX.tif - имя графического файла образа-эталона;

N - число пикселей изображения образа;

NT - число тестируемых образов, полученных путем искажения эталона;

P - доля искаженных пикселей [0; 1].

На рисунках 4.2-4.6 представлены некоторые примеры распознавания символов, изображенных на рис. 4.1, с помощью обученной НС. Обучение проводилось при числе обучающих образов M=10 для каждого вида символа и параметре искажения символов p=0,1.

Рис. 4.2. Неверные распознавания символа «**0**», искаженного 20% шума «Соль и Перец», результат распознавания (слева направо): **«2», «3», «5»**

Рис. 4.3. Правильные распознавания символа «**0**», искаженного 20% шума «Соль и Перец», результат распознавания (слева направо): **«0», «0», «0», «0»**

Рис. 4.4. Неверные распознавания символа «**4**», искаженного 20% шума «Соль и Перец», результат распознавания (слева направо): **«3», «5», «6»**

Рис. 4.5. Правильные распознавания символа «**4**», искаженного 20% шума «Соль и Перец», результат распознавания (слева направо):
«4», «4», «4», «4», «4»

Рис. 4.6. Результаты распознавания символа «**8**», искаженного 10% шума «Соль и Перец», результат распознавания (слева направо):
«8», «8», «6»

Результаты распознавания символов, представленные на рисунках 4.2-4.6, демонстрируют хорошее распознавание с помощью НС даже при сильном искажении (параметр p>0,1). Для объективной оценки качества работы НС необходимо вычисление вероятностных характеристик распознавания. При правильном выборе параметров обучения сети и использовании не менее 100 обучающих образов можно получить вероятность правильного распознавания символов порядка 0,6…0,9 (в зависимости от вида распознаваемого символа) при параметре искажения p=0,1…0,2.

Качество работы НС характеризуется вероятностями правильной классификации образа i-го класса, i=1,…,M. Оценка вероятностей производится по формуле:

$$\hat{P}_{np}(i) = \frac{N_{np}}{N_0},$$

где N_{np} - число правильных распознаваний образа i-го класса; N_0 - общее число распознаваний образов i-го класса. Число определяется экспериментально при запуске программы ***sr_work*** при значениях =10…100.

ЦЕЛЬ РАБОТЫ

Исследование возможностей распознавания печатных символов с помощью нейронных сетей, а также построение нейронных сетей в среде MATLAB.

ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ И ПОРЯДОК ВЫПОЛНЕНИЯ

1. Подготовить графические файлы эталонных образов для символов, заданных преподавателем, или самостоятельная подготовка эталонных (обучающих) образов печатных символов в виде набора графических файлов.

2. В среде MATLAB создать и обучить нейронную сеть (НС), предназначенную для распознавания печатных символов.

3. Исследовать зависимость качества работы НС от:

- степени искажения символов (параметр p);
- числа нейронов в скрытом слое.

ПРИМЕР ВЫПОЛНЕНИЯ РАБОТЫ

Цель работы: обучить нейронную сеть распознавать рукописные цифры.

Перед нами стоит задача классификации: есть десять эталонных образцов, цифры от 0 до 9.

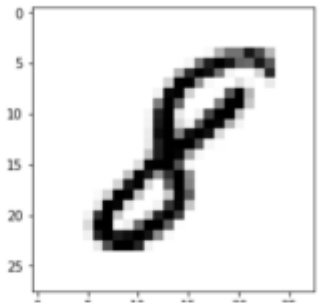
1. Импорт библиотек для работы и загрузка датасета MNIST. MNIST - объёмная датасет образцов рукописного написания цифр. Он включает в себя 60,000 изображений для тренировки and 10,000 тестовых изображений.

In [2]:
```
import ssl

ssl._create_default_https_context = ssl._create_unverified_context

import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

In [24]:
```
from skimage.io import imread, imshow, imsave
from skimage import img_as_float
```

2. Части x_train и x_test содержат изображения в цветовом пространстве RGB в оттенках серого (уровни яркости от 0 до 255). Выведем пример рукописной цифры из датасета MNIST.

```
In [3]: import matplotlib.pyplot as plt
        %matplotlib inline
        image_index = 3434 # Индекс числа из MNIST
        print(y_train[image_index]) # Вывод числа
        plt.imshow(x_train[image_index], cmap='Greys')

        8

Out[3]: <matplotlib.image.AxesImage at 0x1308e2860>
```



3. Тренировочная выборка: 60000 изображений 28*28 пикселей. Тестовая выборка: 10000 изображений 28*28 пикселей.

```
In [4]: x_train.shape

Out[4]: (60000, 28, 28)

In [5]: x_test.shape

Out[5]: (10000, 28, 28)
```

4. Нормализация изображений. Мы должны нормализовать наши данные, как это всегда требуется в моделях нейронных сетей. Мы можем достичь этого, разделив коды RGB на 255 (это максимальное значение для цветового пространства RGB). Преобразуем данные в float. Теперь уровень яркости определяется значением от 0 до 1.

```
In [6]: # Преобразование массива
        x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
        x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
        input_shape = (28, 28, 1)
        # Преобразование float
        x_train = x_train.astype('float32')
        x_test = x_test.astype('float32')
        # Нормализация под цветовое пространство RGB
        x_train /= 255
        x_test /= 255
        print('x_train shape:', x_train.shape)
        print('Number of images in x_train', x_train.shape[0])
        print('Number of images in x_test', x_test.shape[0])

        x_train shape: (60000, 28, 28, 1)
        Number of images in x_train 60000
        Number of images in x_test 10000
```

5. Построение нейронной сети. Мы будем конфигурировать нашу сеть с помощью Kernas API с использованием библиотеки TensorFlow, которая включает инструменты для работы с сетью.

Входные значения: 784 (28*28) значений от 0 до 255 (уровней яркости изображения) На входной слой – 800 нейронов. На выходной слой 10 нейронов и вероятность что на изображении эталонная цифра.

Используется последовательная модель: модель, в которой слои нейронной сети идут друг за другом.

```
In [7]: # Импорт модулей из Keras включ. модели и слои
        from keras.models import Sequential
        from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D

        model = Sequential()
        model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Flatten()) #
        model.add(Dense(128, activation=tf.nn.relu))
        model.add(Dropout(0.2))
        model.add(Dense(10,activation=tf.nn.softmax))

        Using TensorFlow backend.
```

6. Компиляция. Метод обучения – Adam adaptive moment estimation. Он сочетает в себе и идею накопления движения и идею более слабого обновления весов для типичных признаков.

```
In [8]: model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])
        model.fit(x=x_train,y=y_train, epochs=10)
```

7. Запуск (10 эпох) и определение точности.

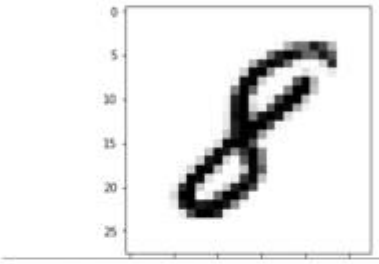
```
Epoch 1/10
60000/60000 [==============================] - 21s 343us/step - loss: 0.2120 - acc: 0.9347
Epoch 2/10
60000/60000 [==============================] - 20s 333us/step - loss: 0.0858 - acc: 0.9734
Epoch 3/10
60000/60000 [==============================] - 20s 333us/step - loss: 0.0582 - acc: 0.9821
Epoch 4/10
60000/60000 [==============================] - 20s 338us/step - loss: 0.0440 - acc: 0.9860
Epoch 5/10
60000/60000 [==============================] - 20s 332us/step - loss: 0.0357 - acc: 0.9880
Epoch 6/10
60000/60000 [==============================] - 21s 345us/step - loss: 0.0290 - acc: 0.9901
Epoch 7/10
60000/60000 [==============================] - 20s 337us/step - loss: 0.0241 - acc: 0.9921
Epoch 8/10
60000/60000 [==============================] - 20s 337us/step - loss: 0.0224 - acc: 0.9922
Epoch 9/10
60000/60000 [==============================] - 20s 341us/step - loss: 0.0213 - acc: 0.9930
Epoch 10/10
60000/60000 [==============================] - 20s 340us/step - loss: 0.0185 - acc: 0.9937
```

Точность модели после сравнения составила 98%.

```
In [9]: model.evaluate(x_test, y_test)

        10000/10000 [==============================] - 1s 133us/step

Out[9]: [0.055517800836151585, 0.9868]
```

8. Проверим индивидуальное распознавание цифр.

```
In [3]: import matplotlib.pyplot as plt
        %matplotlib inline
        image_index = 3434 # Индекс числа из MNIST
        print(y_train[image_index]) # Вывод числа
        plt.imshow(x_train[image_index], cmap='Greys')

        8

Out[3]: <matplotlib.image.AxesImage at 0x1308e2860>
```



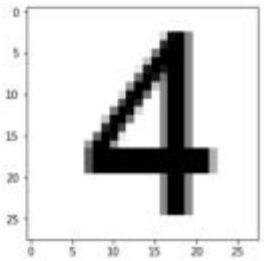
9. Проверим сеть, как она будет работать с искаженными цифрами. Для этого с помощью Adobe Photoshop создаем изображение размером 32*32 и рисуем цифру. Для более оптимизированной работе тестовое изображение сохраняется в формате .tif.

Нормализация изображения для теста.

```
In [76]: test = imread('/Users/thecontrey/Documents/6.tif')
         imshow(test)
         test = test[:,:,1]
         print(test)
         test.shape
```

Без искажений.

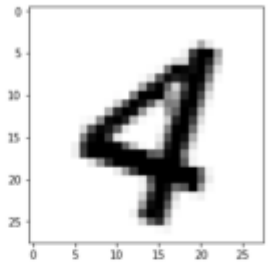
```
In [52]: image = test
         plt.imshow(image.reshape(28, 28),cmap='Greys')
         pred = model.predict(image.reshape(1, 28, 28, 1))
         print(pred.argmax())

         4
```



Фильтр «diffusion» 10% + поворот вправо на 20*.

```
In [56]: image = test
         plt.imshow(image.reshape(28, 28),cmap='Greys')
         pred = model.predict(image.reshape(1, 28, 28, 1))
         print(pred.argmax())
```

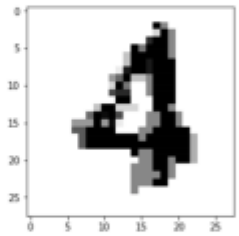
```
4
```



Фильтр «diffusion» 20%.

```
In [54]: image = test
         plt.imshow(image.reshape(28, 28),cmap='Greys')
         pred = model.predict(image.reshape(1, 28, 28, 1))
         print(pred.argmax())
```

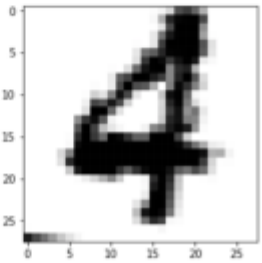
```
2
```



Фильтр «diffusion» 15%, масштабирование, лишний элемент, поворот.

```
In [60]: image = test
         plt.imshow(image.reshape(28, 28),cmap='Greys')
         pred = model.predict(image.reshape(1, 28, 28, 1))
         print(pred.argmax())
```

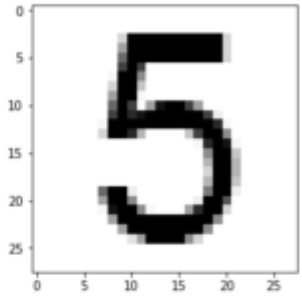
```
6
```



Попробуем другую цифру. Без искажений.

In [66]:
```
image = test
plt.imshow(image.reshape(28, 28),cmap='Greys')
pred = model.predict(image.reshape(1, 28, 28, 1))
print(pred.argmax())
```

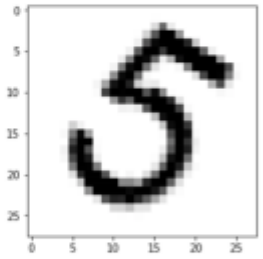
```
5
```



Поворот на 30%.

In [69]:
```
image = test
plt.imshow(image.reshape(28, 28),cmap='Greys')
pred = model.predict(image.reshape(1, 28, 28, 1))
print(pred.argmax())
```

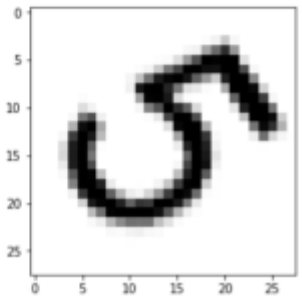
```
5
```



Поворот на 40%.

In [71]:
```
image = test
plt.imshow(image.reshape(28, 28),cmap='Greys')
pred = model.predict(image.reshape(1, 28, 28, 1))
print(pred.argmax())
```

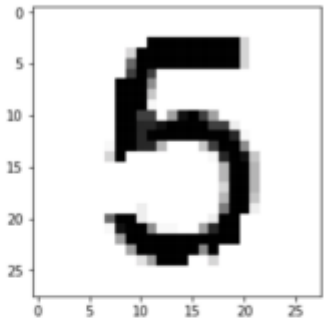
```
0
```



Фильтр «diffusion» 10%.

```
In [73]: image = test
         plt.imshow(image.reshape(28, 28),cmap='Greys')
         pred = model.predict(image.reshape(1, 28, 28, 1))
         print(pred.argmax())
```

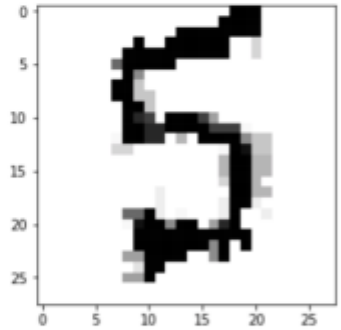
```
5
```



Фильтр «diffusion» 30%.

```
In [75]: image = test
         plt.imshow(image.reshape(28, 28),cmap='Greys')
         pred = model.predict(image.reshape(1, 28, 28, 1))
         print(pred.argmax())
```

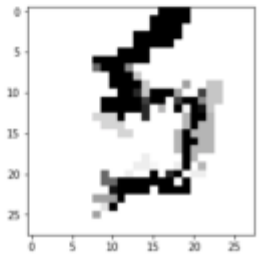
```
5
```



Фильтр «diffusion» 50%.

```
In [77]: image = test
         plt.imshow(image.reshape(28, 28),cmap='Greys')
         pred = model.predict(image.reshape(1, 28, 28, 1))
         print(pred.argmax())
```

```
5
```



Вывод

НС довольно восприимчива к изменениям очертаний символов, особенно на схожих цифрах (например 3, 8 и 9). Однако показывает хорошо на цифрах с более уникальными очертаниями (например 5, 1). Одинаково хорошие результаты получаются при повороте цифр, вплоть до 40 градусов в любую сторону цифры продолжают распознаваться.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назовите основные достоинства и недостатки нейронных сетей?
2. Какие практические задачи решаются с применением нейронных сетей?
3. Назовите негативные последствия переобучения нейронной сети?
4. Дайте характеристику основных этапов построения нейронной сети?