Лабораторная работа №3
ПОСТРОЕНИЕ МОДЕЛИ «ЧЕРНЫЙ ЯЩИК»

Цель работы: получить навыки применения аналитических методов построения модели системы «черный ящик».

1. Модель системы «черный ящик»

Модель "черный ящик" — это концептуальная модель, которая описывает систему, процесс или объект без детализации его внутренней структуры или механизма работы. Важно только то, как система взаимодействует с внешней средой, т.е. какие входные данные она принимает и какие выходные результаты производит.

Модель "черный ящик" основывается на идее абстракции. Она, в основном, применяется в случаях, когда:

- внутренние процессы слишком сложны для полного понимания;
- нет необходимости изучать внутренние механизмы работы системы;
- исследователь или инженер сосредоточены только на входных и выходных параметрах системы.

Если формально обозначить входные данные как X, выходные — как Y, а функцию преобразования как f, то работа черного ящика может быть описана следующим образом:

$$Y=f(X)Y = f(X)Y=f(X),$$

где f — это неизвестная или скрытая функция, которую выполняет черный ящик, преобразующая входные данные в результаты. Внутренние процессы неизвестны или не рассматриваются.

Модель «черного ящика» (см. рис. 1) имеет следующие связи:

- вход (X) — исходные данные (например, сигналы, информация или физические параметры), которые поступают на вход системы;
- выход (Y) — результат работы системы (например, значения, сигналы или конечный продукт);
- обратная связь — наличие схемных циклов в неизменяемой части системы и условных инструкций в её изменяемой части, характеризует зависимость X от Y;
- внешние факторы – описание воздействия на систему внешней среды.

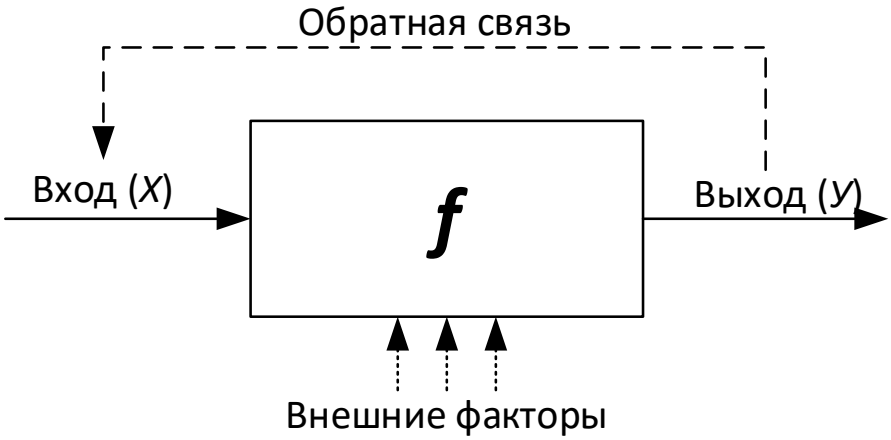

Рисунок 1 – Схема модели системы «черный ящик»

2. Способы построения модели

2.1 Нейронные сети

Нейронные сети – это один из самых известных примеров моделей «черного ящика». Они включают множество слоев с нейронами, которые преобразуют входные данные в выходные путем сложных нелинейных преобразований.

Основные компоненты (рис. 2):

- ***входные слои (Input)*** - принимают данные;
- ***скрытые слои (Hidden)*** - выполняют вычисления (в общем случае это большое количество нелинейных преобразований);
- ***выходной слой (Output)*** - производит результат (классификация или регрессия).

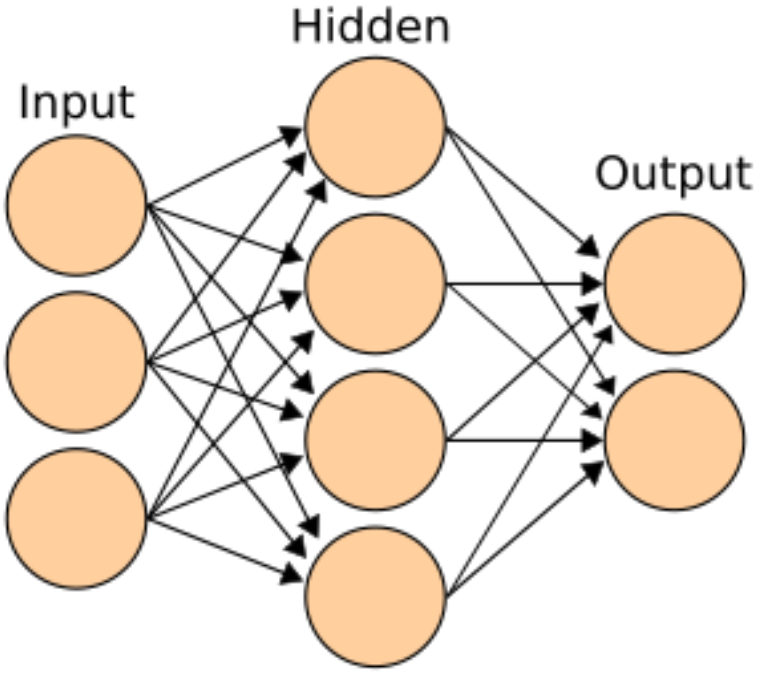

Рисунок 2 – Построение модели «черный ящик» с применением нейронной сети

2.2 Рандомный лес

Рандомный лес – это ансамблевый метод, основанный на использовании множества деревьев решений. Каждый «дерево» принимает решение независимо, а итоговый результат строится на основе голосования всех деревьев.

Основные компоненты (рис. 3):

- *результаты наблюдений (Observation samples)* – исходные данные для решения;
- *множество деревьев решений (Random Forest 1 … Random Forest 3)* - каждое дерево строится на основе случайного набора признаков и случайного набора данных;
- *голосование деревьев (Σ)* - результаты деревьев комбинируются для получения окончательного предсказания;
- *предсказание (prediction)* – результаты расчета модели.

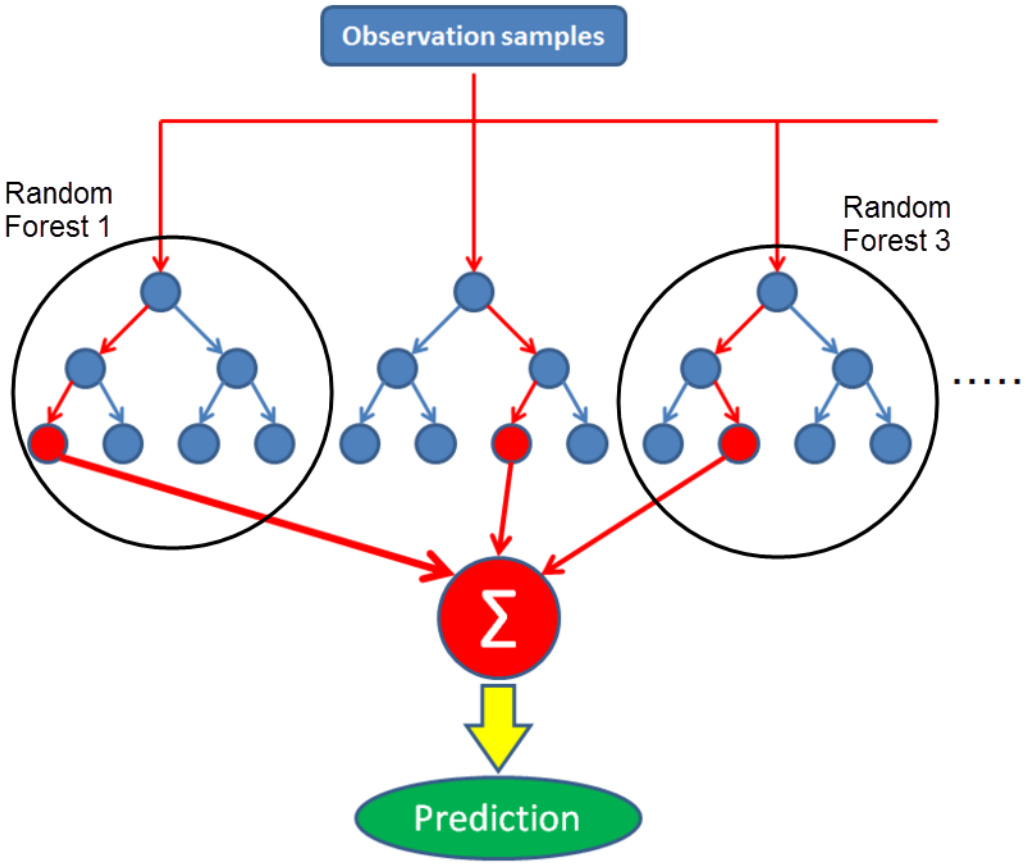

Рисунок 3 – Пример схемы метода Рандомный лес

2.3 Метод опорных векторов (SVM)

Метод опорных векторов (SVM) – это алгоритм классификации, который пытается найти гиперплоскость, максимально разделяющую различные классы. Для нелинейных данных используются ядра, что делает модель более сложной для интерпретации.

Основные компоненты (рис. 4):

- гиперплоскость ($\omega * x - b = 0$) - разделяет классы в пространстве признаков.
- опорные векторы ($\omega * x - b = \pm 1$) - точки данных, наиболее близкие к гиперплоскости и влияющие на её положение.

- ядро – математическая функция для нелинейных данных, необходимая для трансформации данных в более высокое измерение.

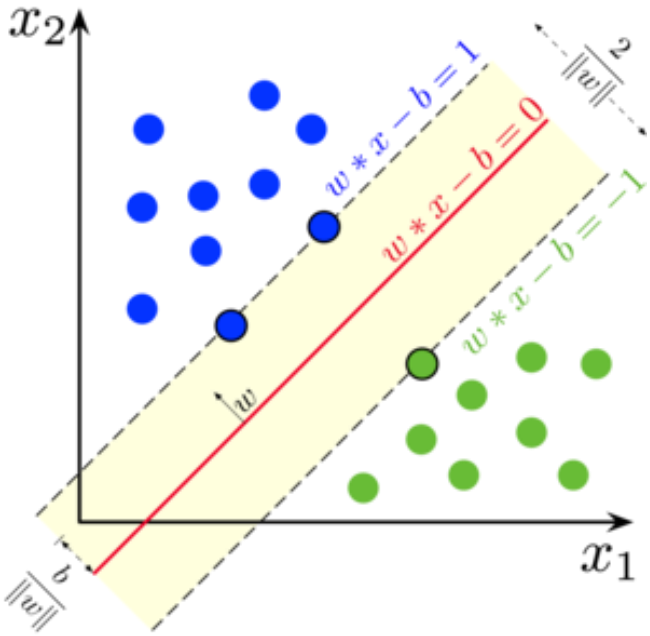

Рисунок 4 - Метод опорных векторов

4. Градиентный бустинг (Gradient Boosting)

Данный метод часто используется в задачах машинного обучения, когда необходима высокая точность предсказаний, но внутреннее устройство модели сложно интерпретировать. Градиентный бустинг строит ансамбль слабых моделей (чаще всего это деревья решений), объединяя их результаты для улучшения прогноза.

Основные этапы построения модели градиентного бустинга (рис. 5):

1) Сбор данных и очистка данных.

2) Инициализация модели. Инициализация слабой модели — первого дерева решений. Предсказание на данном этапе равно среднему значению целевой переменной для регрессии или наиболее частому классу для классификации.

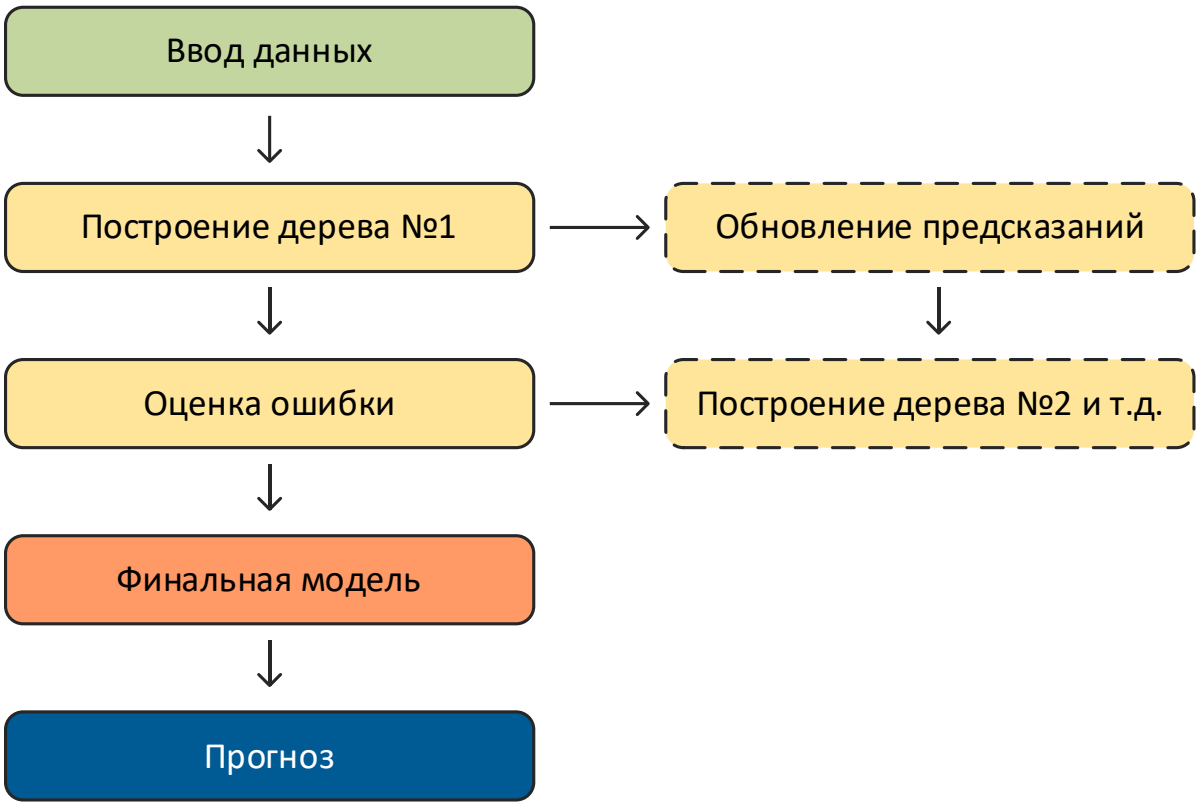

Рисунок 5 – Схема метода Градиентный бустинг

3) Построение ансамбля

а) построение первого дерева: создание слабого классификатора (дерева решений) на основе текущих предсказаний, на выходе формируется «дерево», которое корректирует ошибки предыдущих предсказаний;

б) обновление предсказаний: суммирование результатов предыдущего дерева и нового дерева с учетом скорости обучения;

в) оценка ошибки: вычисление разницы между фактическими значениями и новыми предсказаниями;

4) Повторение шагов для заданного числа деревьев (или до сходимости ошибки).

5) Итоговая модель: ансамбль деревьев решений, которые поэтапно минимизируют ошибки предыдущих предсказаний.

6) Прогнозирование на новых данных.

5. K-ближайших соседей (KNN)

Метод k-ближайших соседей (KNN) основывается на том, что для классификации новых данных модель ищет ближайшие точки (соседей) в пространстве признаков, используя метрики расстояния, такие как Евклидово расстояние. Для каждого нового образца считается расстояние до всех имеющихся данных. Далее выбираются K точек, ближайших к новому образцу, после чего осуществляется классификация на основе большинства соседей.

Рисунок 6 – Схема метода K-ближайших соседей

6. Экстремальные деревья решений (XGBoost)

XGBoost – это одна из реализаций градиентного бустинга, оптимизированная для высокой производительности. Он использует деревья решений и добавляет новую модель на каждом шаге, пытаясь минимизировать ошибку предыдущих шагов. XGBoost оптимизирован для параллельных вычислений.

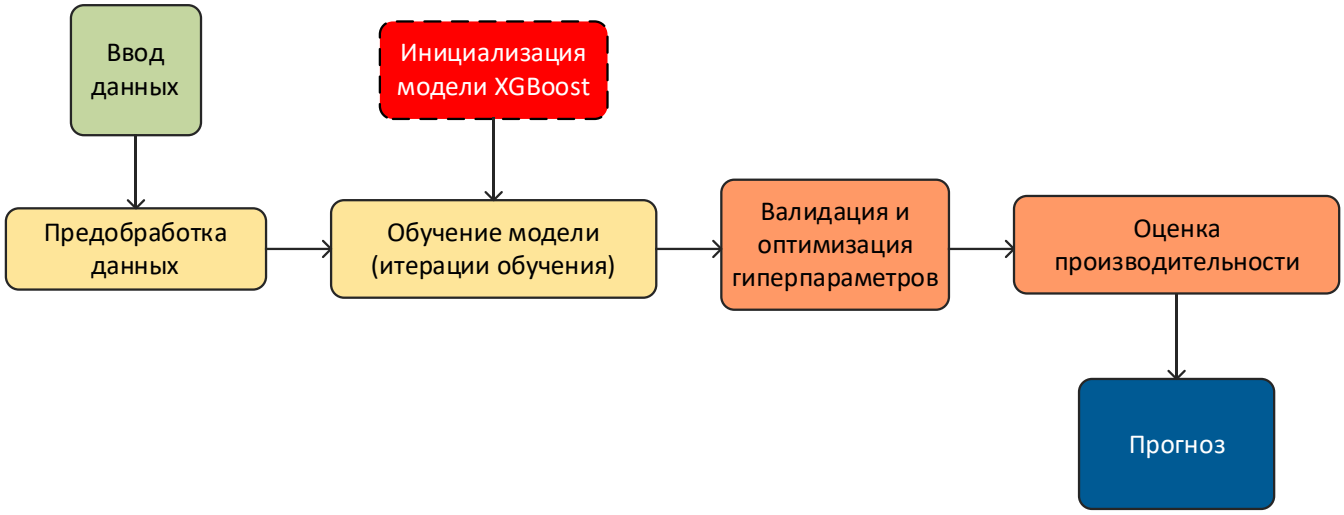

Рисунок 7 – Схема метода XGBoost

Порядок выполнение работы

1. В соответствии с заданием в Jupiter Notebook разработать модель «черного ящика» и рассчитать прогнозные значения минимум одним способом из описанных в теоретическом обосновании.
2. Подготовить отчет о выполнении работы

Содержание отчета

1. Титульный лист
2. Цель работы
3. Номер варианта и задание для выполнения
4. Скриншоты выполнения задания
5. Вывод по работе

6. Приложение – Jupiter Notebook, содержащий задание, краткое описание метода решения и выполнения задания, схему разработанной модели, программный код выполненного задания, результаты прогнозирования и элементы его визуализации.

Полезные ссылки

1. *https://education.yandex.ru/handbook/ml/article/linear-models*
2. *https://education.yandex.ru/handbook/ml/article/gradientnyj-busting*
3. *https://education.yandex.ru/knowledge/metod-opornyh-vektorov.-mashinnoe-obuchenie*
4. *https://education.yandex.ru/handbook/ml/article/metricheskiye-metody*
5. *https://www.kaggle.com/*

Варианты заданий

1. Предсказание времени выполнения задачи разработчика

Предсказать время, необходимое для выполнения задачи разработчиком на основе сложности задачи и опыта разработчика.

№	Сложность задачи (баллы)	Опыт разработчика (лет)	Время выполнения (час)
1	5	1	10
2	3	2	5
3	8	4	12
4	2	5	3
5	7	3	9

2. Предсказание количества найденных багов на основе размера кода

Модель для предсказания количества найденных багов в зависимости от размера исходного кода и числа изменений в коде.

№	Размер кода (строк)	Изменений в коде (раз)	Найдено багов (шт)
1	500	2	3
2	1500	5	10
3	700	1	2
4	2000	6	12
5	1000	3	5

Учесть в модели наличие обратной связи.

3. Оценка трудоемкости тестирования на основе параметров ПО

Предсказать количество часов, необходимых для тестирования, исходя из количества функциональных требований и сложности тестов.

№	Функциональных требований	Сложность тестов (баллы)	Часы тестирования
1	10	3	30
2	20	4	50
3	15	5	60
4	25	2	40
5	30	6	70

Учесть в модели наличие обратной связи.

4. Предсказание производительности ПО в зависимости от параметров системы

Оценка производительности программного обеспечения на разных системах с разными характеристиками.

№	Процессор (ГГц)	ОЗУ (ГБ)	Нагрузка на CPU (%)	Производительность (баллы)
1	2,5	8	50	75
2	3,2	16	30	90
3	1,8	4	70	60
4	2,8	12	40	85
5	3,0	8	60	80

5. Прогноз числа сбоев системы на основе использования ресурсов

Модель прогнозирует количество сбоев системы в зависимости от использования оперативной памяти и процессорных ресурсов.

№	Использование ОЗУ (%)	Нагрузка на CPU (%)	Число сбоев
1	70	50	2
2	80	60	5
3	60	40	1
4	90	70	6
5	50	30	0

Учесть в модели наличие обратной связи.

6. Прогноз времени компиляции на основе параметров проекта

Предсказать время компиляции проекта в зависимости от количества строк кода и числа файлов в проекте.

№	Строк кода	Файлов в проекте	Время компиляции (сек)
1	2000	10	60
2	5000	25	120
3	1000	5	30
4	8000	40	180
5	3000	15	90

Учесть в модели наличие обратной связи.

7. Оценка затрат на разработку в зависимости от продолжительности и команды

Оценить затраты на разработку в зависимости от продолжительности проекта и количества разработчиков.

№	Продолжительность (мес)	Кол-во разработчиков	Затраты (тыс. $)
1	6	5	300
2	12	8	800
3	3	3	100
4	9	7	500
5	15	10	1000

Учесть в модели наличие двойной обратной связи.

8. Прогноз использования оперативной памяти на основе характеристик данных

Модель для предсказания использования оперативной памяти программой в зависимости от объема данных и сложности обработки.

№	Объем данных (ГБ)	Сложность обработки (баллы)	Использование ОЗУ (ГБ)
1	5	3	8
2	10	4	16
3	7	5	12
4	12	2	20
5	15	6	24

Учесть в модели наличие двойной обратной связи.

9. Предсказание числа изменений кода в зависимости от числа багов

Предсказать количество изменений в коде после обнаружения багов.

№	Найдено багов	Количество изменений
1	5	10
2	10	20
3	3	5
4	8	15
5	15	25

10. Предсказание времени загрузки приложения на основе размера

Предсказать время загрузки приложения в зависимости от его размера и числа библиотек.

№	Размер приложения (МБ)	Количество библиотек	Время загрузки (сек)
1	50	10	5
2	100	20	10
3	75	15	8
4	150	25	15
5	200	30	20

Учесть в модели наличие обратной связи.

11. Прогнозируемая успешность релиза по числу тестов и багов

Прогноз успешности релиза в зависимости от количества пройденных тестов и оставшихся багов.

№	Пройдено тестов	Оставшиеся баги	Успешность релиза (баллы)
1	100	5	85
2	200	10	80
3	150	3	90
4	250	15	70
5	300	20	65

12. Прогноз времени сборки проекта на основе числа задач

Оценка времени сборки проекта в зависимости от количества завершенных задач и объема кода.

№	Завершено задач	Строк кода	Время сборки (мин)
1	20	2000	15
2	30	3000	20
3	15	1000	10
4	40	5000	30
5	50	6000	35

Учесть в модели наличие обратной связи.

13. Прогноз времени развертывания ПО на основе инфраструктуры

Предсказать время развертывания программного обеспечения на основе параметров инфраструктуры.

№	Серверов	Виртуальных машин	Время развертывания (мин)
1	2	5	10
2	4	10	20
3	6	15	30
4	8	20	40
5	10	25	50

Учесть в модели наличие двойной обратной связи.

14. Прогноз качества кода на основе метрик качества

Оценить качество кода на основе количества комментариев, цикломатической сложности и глубины вложенности.

№	Комментариев (%)	Цикломатическая сложность	Глубина вложенности	Качество кода (баллы)
1	10	15	3	70
2	20	10	2	80
3	15	20	4	60
4	25	8	1	90
5	5	25	5	50

Учесть в модели наличие обратной связи.

15. Предсказание времени релиза по числу закрытых задач

Оценить время релиза проекта на основе числа закрытых задач и оставшихся открытых задач.

№	Закрыто задач	Открыто задач	Время до релиза (недели)
1	50	10	3
2	100	20	6
3	70	15	5
4	150	25	8
5	200	30	10

Учесть в модели наличие обратной связи.

16. Прогноз нагрузки сервера в зависимости от числа запросов

Оценить нагрузку на сервер в зависимости от числа обрабатываемых запросов и сложности запроса.

№	Число запросов	Сложность запросов (баллы)	Нагрузка на сервер (%)
1	1000	3	40
2	2000	4	60
3	500	2	20
4	3000	5	80
5	4000	6	90

17. Прогноз времени на исправление багов в зависимости от их сложности

Оценить время, необходимое на исправление багов в зависимости от их сложности и количества.

№	Сложность бага (баллы)	Количество багов	Время исправления (час)
1	3	5	10
2	4	8	16
3	2	3	6
4	5	10	20
5	6	12	24

18. Предсказание времени работы ПО до сбоя на основе нагрузки

Оценить время работы ПО до сбоя на основе нагрузки на процессор и оперативную память.

№	Нагрузка на CPU (%)	Использование ОЗУ (%)	Время работы до сбоя (час)
1	50	70	24
2	60	80	18
3	40	60	30
4	70	90	12
5	80	95	8

19. Прогноз нагрузки на сеть в зависимости от числа пользователей

Оценить нагрузку на сеть в зависимости от количества активных пользователей и числа запросов.

№	Активных пользователей	Запросов на пользователя	Нагрузка на сеть (%)
1	500	10	40
2	1000	20	70
3	300	15	30
4	2000	25	90
5	1500	18	80

Учесть в модели наличие двойной обратной связи.

20. Прогноз числа необходимых тестов для покрытия кода

Оценить количество тестов, необходимых для полного покрытия кода, в зависимости от количества строк кода и сложности программы.

№	Строк кода	Сложность программы (баллы)	Количество тестов
1	2000	3	50
2	5000	4	100
3	3000	2	60
4	10000	5	150
5	8000	6	200