ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
**Московский институт электроники и математики**

# Моя первая нейронная сеть. Классификация графов с помощью НС

Проектная работа студента образовательной программы бакалавриата
«Прикладная математика»
по направлению подготовки 01.03.04 ПРИКЛАДНАЯ МАТЕМАТИКА

Студент
Морозов Д.С.

Руководитель
Профессор
В. Ю. Попов

**Москва 2024г.**

# Аннотация

В данной работе рассматривается применение нейросетевых алгоритмов для решения задачи определения гамильтоновости графа. Цель проекта — создать модель машинного обучения, способную эффективно и точно определять наличие гамильтоновых циклов в графах различной структуры и размерности.

Этапы работы включают в себя обзор теоретических условий гамильтоновости, генерацию датасетов различного размера и непосредственную работу с нейросетями. Описывается процесс преобразования графа в эмбеддинги, обучение нескольких моделей, их оптимизацию и валидацию. В конце приводится анализ применимости такого подхода.

Ожидается, что результаты проекта найдут применение в различных областях, где требуется быстрое определение гамильтоновых путей и циклов, например, в задачах оптимизации маршрутов, планирования и анализа сетей. Проект также может способствовать развитию теоретических аспектов теории графов и машинного обучения.

# Содержание

# Введение

В современной науке гамильтоновы графы представляют практический интерес для изучения и моделирования конкретных задач. [1] Например, нерешенная задача о коммивояжере является частным случаем гамильтоновости графа.

Ключевой задачей в теории алгоритмов является решенение так называемых NP-полных задач. Множество задач P характеризуется существованием алгоритма решение, время работы которого полиномиально зависит от размера входных данных, $O(n^p)$. В NP-задачах стоит вопрос о принадлежности задачи к классу P и разрешимости их за полиномиальное время. Задача называется NP-полной, если ее принадлежность к классу P означает принадлежность к этом классу все остальных NP-задач. В частности, определение гамильтоновости графа является одной из нерешенных NP-полных задач.

Гамильтоновы графы используются в криптографии, в качестве хеш-функции для протоколов с нулевым разглашением. При этом сам граф является объектом шифрования, а гамильтонов цикл ключом, который можно передать в зашифрованном виде для изоморфного графа, не раскрывая при этом изначального графа.

В квантовых вычислениях гамильтоновы графы моделируют непосредственно кубиты информации в квантовом компьютере и взаимоотношения между ними. Применение знаний о гамильтоновости графа позволит ускорить сложнопроизводительные задачи и увеличить эффективность процессоров.

Также гамильтоновы графы используются в задачах логистики, в моделировании углеродных нанотрубок стабильной конфигурации, в системах автоматического проектирования в инженерии и моделировании химических реакций.

В данный момент не существует эффективного алгоритма для определения гамильтоновости графа. [2] Используемые на практике методы в большинстве основаны на алгоритме brute-force, то есть полного перебора, что для графа составляет сложность $O(n!)$. Граф, состоящий примерно из 30 вершин и большого количества ребер может проверяться системой около получаса, что крайне долго для практических задач. Поэтому целью работы является оптимизация классификации графов по гамильтоновости с помощью использования нейросетей и глубинного обу-

чения.

**Цель работы** – разработать нейросетевую модель, которая сможет определять гамильтоновость графов с точностью не менее 95 %.

**Задачи:**

- Обзор методов работы с графами
- Генерация датасетов гамильтоновости графов
- Преобразование графов в эмбеддинги
- Обучение моделей машинного обучения и MLP
- Отбор моделей и подбор гиперпараметров
- Анализ полученных результатов и путей дальнейшего развития проекта

# 1 О теории графов

## 1.1 Свойства гамильтоновости графа

Графом $G(V, E)$ будем называть совокупность пар вершин и ребер без петель и кратных ребер. Если в графе имеется простой остовный цикл, то граф называется гамильтоновым.[3] Для классификации имеет смысл рассматривать только связные графы.

**Достаточное условие гамильтоновости графа (теорема Дирака)**: Если в графе $G(V, E)$ с $p \geq 3$ вершинами $\forall v \in V$ степень $v \geq p/2$, то граф G является гамильтоновым. [4]

**Теорема Оре**: Если в графе $G$ для любой пары несмежных вершин $v$ и $w$ их степень вершин будет большей или равной количеству вершин в графе, то G является гамильтоновым.

Будем рассматривать две выборки гамильтоновых графов: полную и выборочную. В первой учтем графы с любым количеством ребер от $v - 1$ до $\frac{v(v-1)}{2}$, множество графов оттуда будут отсечены достаточными условиями выше. В выборочной

выборке будут находиться графы с количеством ребер не более $\frac{v^2}{4}$, что поможет изучить конкретно интересующую область графов.

## 1.2 Представление графов в памяти компьютера

В рамках данной работы для анализа и манипуляции структурами графов на языке Python будет применяться библиотека NetworkX. Стандартно графы храняться в библиотеке как отдельный класс, содержащий неориентированные ребра. Рассмотрим несколько подходов к обработке графов с их преимуществами и недостатками и выберем подходящий для наших целей:

### 1.2.1 Матрица смежности

Для хранения рёбер используется двумерная матрица размерности $[V, V]$, каждый $[a, b]$ элемент которой равен 1, если вершины $a$ и $b$ являются смежными и 0 в противном случае. [5]

В случае неориентированного графа матрица является симметричной относительно главной диагонали, а сумма каждой строчки и каждого столбца равна степени вершины.

Представив граф в виде матрицы смежности можно преобразовать ее в многомерный тензор и обучить на этом CNN модели, как будто работа идет с картинкой.

Неоднозначность такого подхода в том, что среднестатистические модели предобучены на стандартных датасетах по типу ImageNet, которые содержат реально значащие картинки. Матрица смежности же будет бинарным изображением с хаотическим расположением черных и белых пикселей.

### 1.2.2 Матрица признаков

Для работы с данными при помощи моделей машинного обучения необходимы табличные данные. Можно сделать выделение признаков графов вручную, например, количество ребер, средняя степень вершины, коэффицент Жаккара, планарность, хроматическое число графа, вырожденность матрицы смежности и так далее.

Преимуществом такого подхода будет интерпретируемость данных, можно будет сделать некоторые эвристические выводы о признаках и зависимости их на гамильтоновость графа. Основным же недостатком является трудозатратность генерации таких признаков вручную, а также сложности в извлечении некоторых из них из графов. Например, проверка планарности графа требует асимптотической сложности $O(n)$, что довольно немного, однако при большом количестве таких признаков извлечение признаков может занимать значительное время.

### 1.2.3 Эмбеддинги

Нам бы хотелось оптимизировать и автоматизировать процесс извлечения признаков из графов с использованием нейронной сети.

**Формулировка задачи**: Дан набор графов $\mathbb{G} = \{G1, G2, \dots\}$ и целое положительное число $\delta$ (ожидаемый embedding size), необходимо получить $\delta$-мерные векторы для представления каждого графа $G_i \in \mathbb{G}$.

В статье [6] рассматривается аналогия между нейросетью для выделения эмбеддингов word2vec, используемой в NLP (Natural Language Processing) и graph2vec.

Принцип работы word2vec строится на предположении, что слова, встречающиеся в схожих контекстах, как правило, имеют схожие семантические значения и, следовательно, должны иметь схожие векторные представления. Чтобы узнать эмбеддинг целевого слова, эта модель использует понятие контекста – фиксированного количества слов, окружающих целевое слово.

Архитектура нейронной сети skipgram выделяет эмбеддинги на основе контекста целевого слова и максимизирует следующую логарифмическую вероятность: [7]

$$\sum_{t=1}^{T} \log Pr(w_{t-c}, \dots, w_{t+c} | w_t), \tag{1}$$

где $w_t$ - целевое слово, $c$ - расстояние окна контекста слова, $w_{t-c}, \dots, w_{t+c}$ - контекст слова.

Вероятность $Pr(w_{t-c}, \dots, w_{t+c} | w_t)$ вычисляется по формулам 2 и 3

$$\prod_{-c \leq l \leq c, j \neq 0} Pr(w_{t+j}|w_t) \tag{2}$$

$$\frac{\exp(\vec{w_t} \cdot \vec{w'_{t+j}})}{\sum_{w \in \nu} \exp(\vec{w_t} \cdot \vec{w}),} \tag{3}$$

$\nu$ - словарь всех слов, $\vec{w}$ и $\vec{w'}$ - входные и выходные векторы слова $w$

**Doc2vec** – прямое дополнение к word2vec, позволяющее изучать вложения слов в различные последовательности слов. Doc2vec также использует модель skipgram, которая может обучаться представления последовательностей слов произвольной длины (предложения, абзацы и даже целые большие документы)

Переходя от модели обработки документов к графам, будем использовать предположение, что весь граф рассматривается как документ, а корневые подграфы вокруг каждой вершины в графе как слова, составляющие документ. Другими словами, разные подграфы образуют графы аналогично тому, как разные слова образуют предложения или документы.

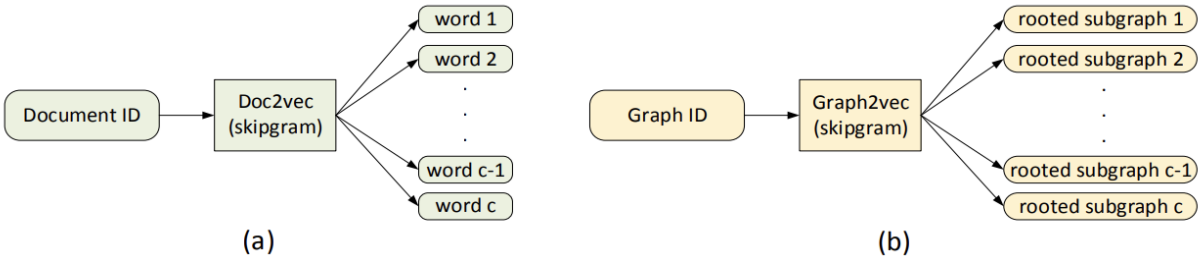


Рис. 1: Аналогия между doc2vec и graph2vec

Установив вышеупомянутую аналогию документов и слов с графами и корневыми подграфами, можно использовать нейронную архитектуру skipgram и для графов. Основное предположение заключается в том, что структурно похожие графы будут близки друг к другу в векторном виде.

Имея датасет из графов, graph2vec рассматривает набор всех корневых подграфов (т.е. окрестностей вершин) вокруг каждой вершины (до определенной степени $c$) в качестве словаря. Далее выполняется процесс обучения skipgram с помощью doc2vec, по приведенным выше формулам.

**Алгоритм** состоит из двух основных компонентов:

1. процедура генерации корневых подграфов вокруг каждой вершины в данном графе

2. процедура обучения эмбеддингов данных графов

Как показано на рис.2 и 3, мы хотим получить $\delta$-мерные эмбеддинги для всех графов в датасете $\mathbb{G}$ за $e$ эпох. Начиная со случайной инициализации эмбеддингов для всех графов в датасете (строка 2), приступаем к извлечению корневых подграфов вокруг каждой вершины для каждого из графов (строка 8) и итеративно обучаем (т.е. уточняем) эмбеддинг соответствующего графа в несколько эпох (строки с 3 по 10).

**Algorithm 1:** GRAPH2VEC $(\mathbb{G}, D, \delta, \mathfrak{e}, \alpha)$

**input :** $\mathbb{G} = \{G_1, G_2, ..., G_n\}$: Set of graphs such that each graph $G_i = (N_i, E_i, \lambda_i)$ for which embeddings have to be learnt
$D$: Maximum degree of rooted subgraphs to be considered for learning embeddings. This will produce a vocabulary of subgraphs, $SG_{vocab} = \{sg_1, sg_2, ...\}$ from all the graphs in $\mathbb{G}$
$\delta$: number of dimensions (embedding size)
$\mathfrak{e}$: number of epochs
$\alpha$: Learning rate

**output:** Matrix of vector representations of graphs $\Phi \in \mathbb{R}^{|\mathbb{G}| \times \delta}$

1 **begin**
2  Initialization: Sample $\Phi$ from $R^{|\mathbb{G}| \times \delta}$
3  **for** $e = 1$ *to* $\mathfrak{e}$ **do**
4  $\mathfrak{G} =$ SHUFFLE $(\mathbb{G})$
5  **for** ***each*** $G_i \in \mathfrak{G}$ **do**
6  **for** ***each*** $n \in N_i$ **do**
7  **for** $d = 0$ *to* $D$ **do**
8  $sg_n^{(d)} :=$ GETWLSUBGRAPH$(n, G_i, d)$
9  $J(\Phi) = -\log \Pr\ (sg_n^{(d)} | \Phi(G))$
10  $\Phi = \Phi - \alpha \frac{\partial J}{\partial \Phi}$
11  **return** $\Phi$

Рис. 2: Алгоритм graph2vec в псевдокоде

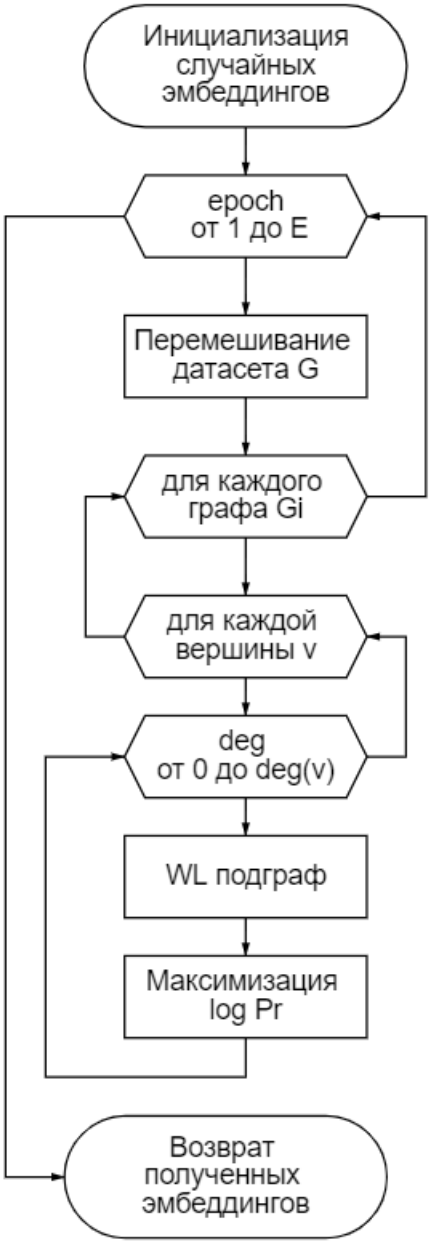


Рис. 3: Алгоритм graph2vec в блок-схеме

# 2 Генерация датасета

Для создания датасета необходима возможность генерации случайных связных графов и проверки их на гамильтоновость. Такой алгоритм вшит в систему Maple 2024 в пакетах $RandomGraphs$ и $GraphTheory$.

```
printGraph := proc()
local V, mxE, E, G, bool, is_hamilton;
V := 15;
mxE := 1/2*V*(V - 1);
E := (rand() mod (mxE - V + 2)) + V - 1;
G := RandomGraph(V, E, connected);
bool := IsHamiltonian(G);
ExportGraph(G, "hamilton", graph6, append);
return subs([false = 0, true = 1], bool);
end proc;
```

Рис. 4: Генерация связного графа на 15 вершинах

```
main := proc()
local target, i;
target := String();
for i from 0 to 1000 do
target := cat(target, printGraph());
end do;
Export("bools.txt", target);
end proc;
```

Рис. 5: Формирование датасета из 1000 элементов в отдельном файле

Время генерации возрастает экспоненциально. Например, датасет на 10000 графов генерируется 3911.81 секунд, а на 50000 графов более суток. Также сами графы проходят проверку на гамильтоновость неравномерно - некоторые могут отсеяться по достаточному признаку в доли секунд, а другие брутфорсится около получаса.

Именно поэтому эффективнее для большей однородности датасета использовать выборочную генерацию, несмотря на то, что она и дольше (10000 графов на 20 вершин - 50525.93 секунды)

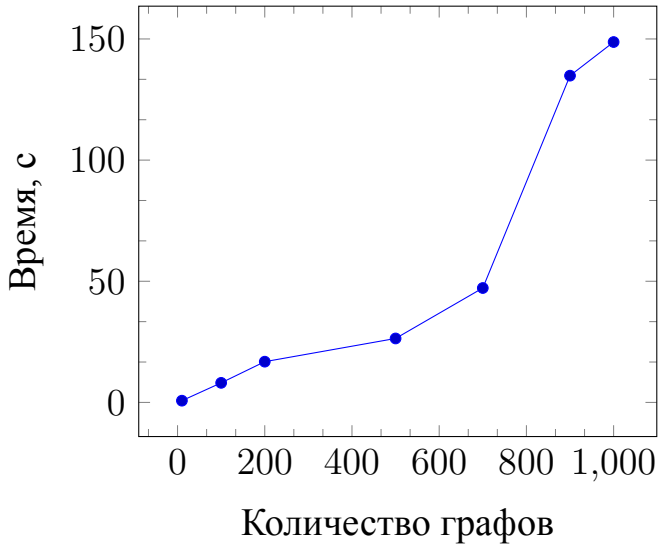


Рис. 6: Зависимость времени генерации графов от их количества

# 3 Баланс классов

Как ключевая метрика будет использована accuracy, значит крайне важен баланс классов. При генерации связных графов с количеством ребер от $v-1$ до $\frac{v(v-1)}{2}$ в большинстве случаев они получаются гамильтоновыми, исходя из теорем Дирака и Оре.

Действительно, доля положительного класса в полном датасете значительно больше, чем выборочном, как видно на рисунках 7 и .8

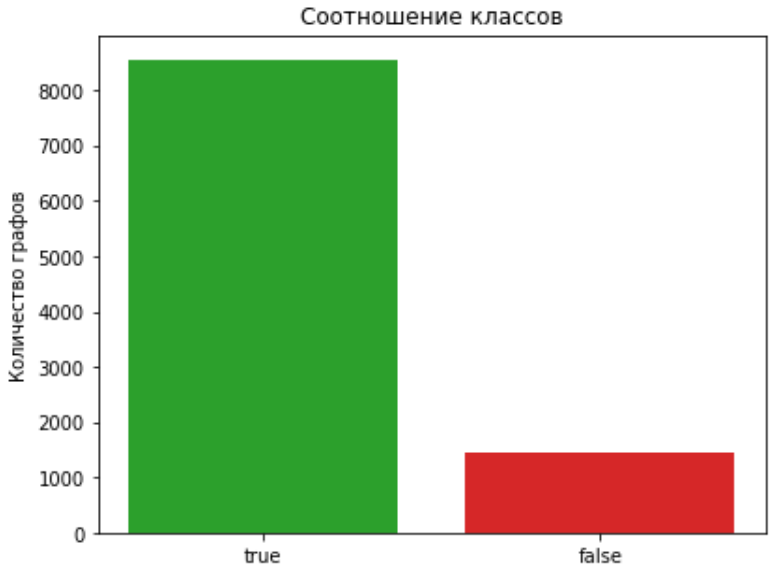


Рис. 7: Распределение классов для графов с 20 вершинами - полный

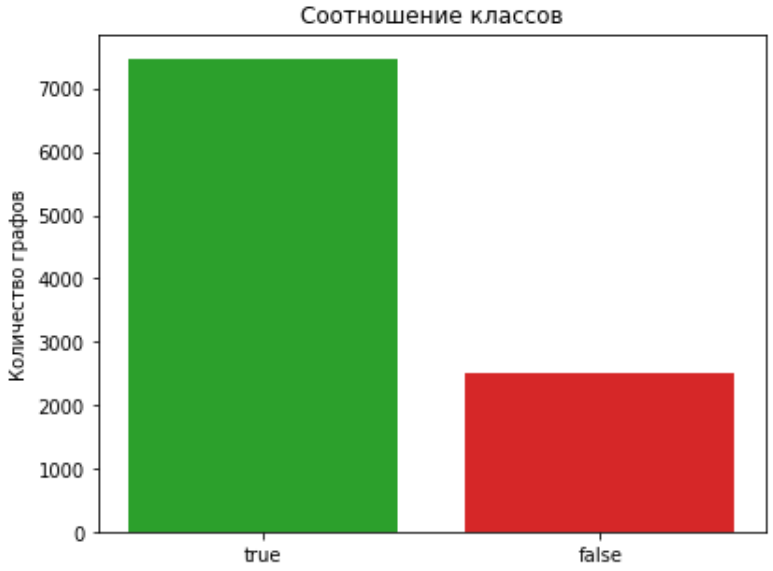


Рис. 8: Распределение классов для графов с 20 вершинами - выборочный

Удаление значительной части положительного класса не критично, так как есть возможность заранее сгенерировать датасет любой необходимой величины.

# 4 Выделение эмбеддингов

Для выделения эмбеддингов из графов использовался репозиторий graph2vec.[8] Он написан на python 2.7 со старыми версиями библиотек, поэтому пришлось внести некоторые необходимые изменения для его работоспособности [Измененная версия кода].

Для обучения датасет необходимо привести в подходящий формат: каждый граф должен храниться в формате «i.gexf», где $i$ - порядок соответсвующего графа в специальной папке. Метки target должны быть записаны в файл расширения .Labels с указанием названия графового файла, которому каждая из них соответствует. После проведения данной процеды можно делать запуск файла «main.py», указав папку графов, файл лейблов и различные гиперпараметры.

В таблице 1 указаны результаты обучения на соответствующих датасетах. Файлы с обученными эмбеддингами хранятся на диске.

| кол-во вершин | кол-во графов | тип | кол-во эпох | размер эмбеддинга | learning rate |
|---|---|---|---|---|---|
| 15 | 1000 | full | 1000 | 1024 | 0.3 |
| 15 | 1000 | full | 1000 | 128 | 0.2 |
| 20 | 10000 | full | 1000 | 128 | 0.2 |
| 20 | 10000 | selected | 1000 | 128 | 0.2 |
| 20 | 10000 | selected | 500 | 1024 | 0.1 |

Таблица 1: Обученные датасеты эмбеддингов

# 5 Обучение моделей

Для применения моделей машинного обучения необходимо разделить все данные на трейн и тест. Однако для этого предварительно нужно расписать эмбеддинги и целевые значения в массивы $X$ и $y$ соответственно, а также сделать масштабирование данных. Поскольку мы собираемся использовать SVC и MLP в качестве моделей, то от нормализации данных напрямую зависит скорость сходимости градиентного спуска. Для масштабирования используем MinMaxScaler из библиотеки

sklearn, приводящий все данные в диапазон [0, 1]. [9]

При разделении на трейн и тест стратификацию использовать нет необходимости, так как классы уже сбалансированы. Объем тестовой выборки берем как $15\%$.

**Модели, выбранные для обучения:**

- KNN
- Наивный Байесовский классификатор
- Random Forest
- SVC
- SGD
- Градиентный бустинг

Это стандартная выборка моделей, содержащая базовые алгоритмы машинного обучения, реализованные в библиотеке sklearn.

**KNN** – модель, классифицирующая целевой объект, исходя из того, какие классы у объектов, которые максимально похожи на него (близки по расстоянию). Метрика расстояния - евклидова норма. Настраеваемым гиперпараметром возьмем количество соседей. В нашем случае его использование логически обоснованно тем, что похожие графы имеют схожие векторные представления и скорее всего совпадают по гамильтоновости.[10]

**Наивный Баесовский классификатор** – вероятностный классификатор, использующий для оценки плотности многомерного распределения $P(X|Y)$ оценки плотности одномерных распределений $P(X^i|Y)$ при предположении, что все признаки линейно независимы:

$$P(X|Y) = P(X^1, X^2, \ldots, X^d|Y) = P(X^1|Y)P(X^2|Y)\ldots P(X^d|Y)$$

Для оценки используем Гауссовский наивный байесовский классификатор, потому что признаки, с которыми мы работаем непрерывны, лежат в отрезке [0,1] и

скорее всего имеют гауссовское распределение. Настраеваемый гиперпараметр - наибольшая дисперсии из всех фич, которая добавляется к дисперсиям для обеспечения стабильности расчета (var_smoothing)

**Random Forest** – ансамблиевый метод обучения, основанный на одновременном обучение различных решающих деревьев с со случайным набором признаков. Итоговым прогнозом считается самый частовстречающийся из деревьев класс. Гиперпараметром настройки служит количество деревьев в лесу.

**SVC (метод опорных векторов)** – алгоритм, основанный на поиске гиперплоскости или линии, разделяющей классы наилучшим образом. Расположенные ближе всего к разделяющей гиперплоскости эмбеддинги называются опорными векторами. Мы используем линейную версию SVC, потому что она минимизирует квадрат от функции потерь hinge, что работает быстрее, чем при указания линейности ядра путем SVC(kernel=”linear”). Настраеваемый гиперпараметр – коэффицент регуляризации C.

**SGD** – подвид алгоритма градентного спуска, подгоняющий линейные классификаторы под выпуклые функции потерь, такие как линейный SVC и логистическая регрессия. SGD реализует процедуру стохастического спуска, где шаг делается не относительно всех объектов, а определенной малой выборки. Основные параметры модели взяты стандартными, настраивается коэффицент регуляризации alpha.

**Градиентный бустинг** – ансамблиевый алгоритм, последовательно улучшающий показания слабых классификаторов (решающих деревьев) и минимизирующий функцию потерь MSE с помощью градиентного спуска. Данный алгоритм реализован наиболее эффективно в библиотеке xgboost. Настраиваемый гиперпараметр - коэффицент gamma, контролирующий тенденцию модели к переобучению.

По итогам обучения лучше всего проявила себя модель LinearSVC, показав точность в 95.2%. Остальные результаты можно посмотреть в таблице 2 и на рис.9.

| Модель | Гиперпараметр | Значение | Точность |
|---|---|---|---|
| KNN | n_neighbors | 25 | 87.1% |
| NaiveBayes | var_smoothing | 0.12 | 93.5% |
| Random Forest | n_estimators | 1000 | 93.9% |
| SVC | C | 0.1 | 95.2% |
| SGD | alpha | 0.01 | 94.8% |
| Gradient Boost | gamma | 0.01 | 94.7% |

Таблица 2: Результаты скоринга моделей и подбора гиперпараметров

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
Лучшиие гиперпараметры {'n_neighbors': 25}
KNeighbours accuracy:0.8718626155878467
              precision    recall  f1-score   support

           0       0.81      0.96      0.88       356
           1       0.96      0.80      0.87       401

    accuracy                           0.87       757
   macro avg       0.88      0.88      0.87       757
weighted avg       0.89      0.87      0.87       757

Fitting 5 folds for each of 100 candidates, totalling 500 fits
Лучшиие гиперпараметры {'var_smoothing': 0.12328467394420659}
NaiveBayes accuracy:0.9352708058124174
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       356
           1       0.94      0.93      0.94       401

    accuracy                           0.94       757
   macro avg       0.93      0.94      0.94       757
weighted avg       0.94      0.94      0.94       757

Fitting 5 folds for each of 7 candidates, totalling 35 fits
Лучшиие гиперпараметры {'n_estimators': 1000}
RandomForest accuracy:0.9392338177014531
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       356
           1       0.94      0.94      0.94       401

    accuracy                           0.94       757
   macro avg       0.94      0.94      0.94       757
weighted avg       0.94      0.94      0.94       757
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Лучшиие гиперпараметры {'C': 0.1}
LinearSVC accuracy:0.952443857331572
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       356
           1       0.96      0.95      0.95       401

    accuracy                           0.95       757
   macro avg       0.95      0.95      0.95       757
weighted avg       0.95      0.95      0.95       757

Fitting 5 folds for each of 6 candidates, totalling 30 fits
Лучшиие гиперпараметры {'alpha': 0.01}
SGD accuracy:0.9484808454425363
              precision    recall  f1-score   support

           0       0.95      0.94      0.94       356
           1       0.95      0.96      0.95       401

    accuracy                           0.95       757
   macro avg       0.95      0.95      0.95       757
weighted avg       0.95      0.95      0.95       757

Fitting 5 folds for each of 6 candidates, totalling 30 fits
Лучшиие гиперпараметры {'gamma': 0.01}
GradientBoost accuracy:0.9471598414795245
              precision    recall  f1-score   support

           0       0.94      0.95      0.94       356
           1       0.95      0.95      0.95       401

    accuracy                           0.95       757
   macro avg       0.95      0.95      0.95       757
weighted avg       0.95      0.95      0.95       757
```

Рис. 9: Результаты метрик классификаторов

# 6 MLP

**Многослойный персептрон** – нейросетевой алгоритм, обучающий функцию $f : R^m \rightarrow R^o$, где m — количество измерений для ввода и o - количество размеров для вывода. Ключевым отличием многослойного персептрона от других моделей является наличие между входным и выходным слоями одного или нескольких нелинейных слоев, называемых скрытыми слоями, рис. 10. [11]

Модель многослойного персептрона использует back-propagation или же алгоритм обратного распространения, использующийся для настройки весов между нейронами. Сначала совершается прямой проход forward pass и вычисление всех промежуточных представлений. После чего делается обратный проход backward pass, на котором вычисляются все градиенты. С помощью полученных градиентов происходит шаг в градиентном спуске. Таким образом ошибка передается от выходного слоя к входному для коррекции весов нейронов.

Самый левый (входной) слой состоит из набора нейронов $x_i|x_1, x_2, \ldots, x_m$ представляющие входные функции. Каждый нейрон в скрытом слое преобразует значения из предыдущего слоя с помощью линейной комбинации: $w_1x_1 + w_2x_2 + \ldots + w_mx_m$, за которой следует нелинейная функция активации. Выходной слой получает значения из последнего скрытого слоя и преобразует их в соответствующие выходные значения.

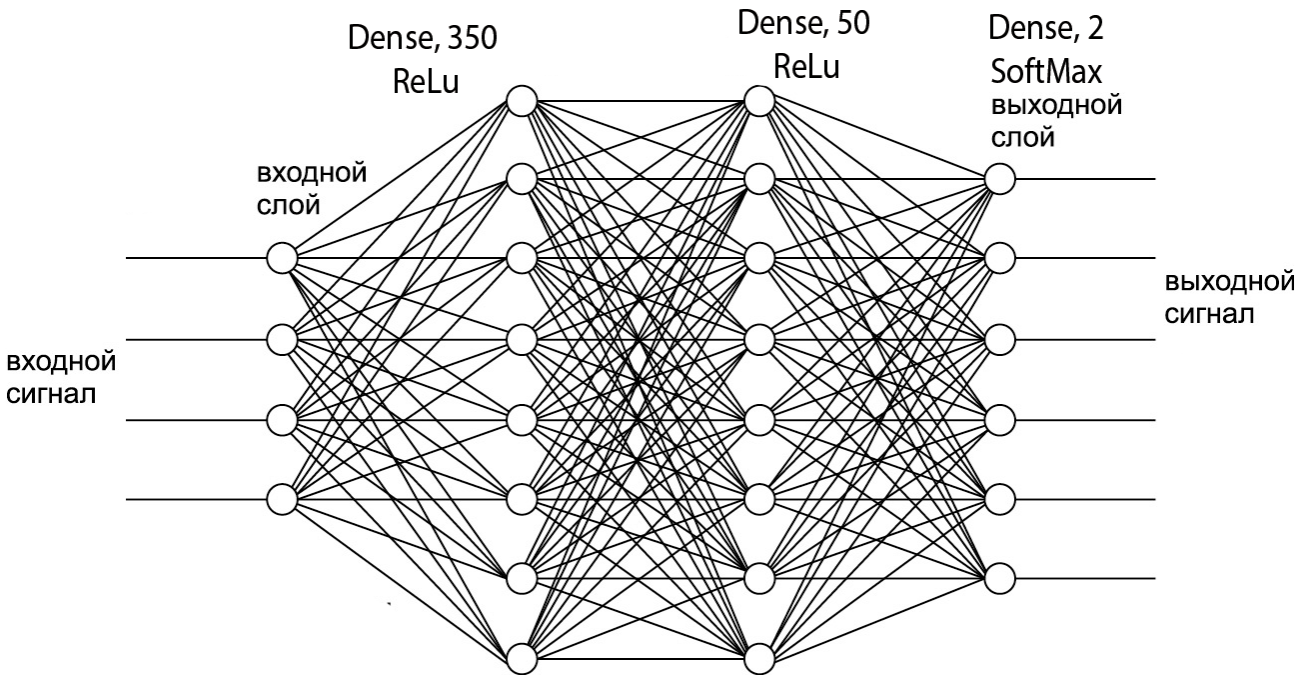


Рис. 10: Модель многослойного персептрона

В нашем случае персептрон имеет три слоя: Dense размера (None, 350) с режимом активации ReLu, Dense размера (None, 50) с режимом активации ReLu и Dense размера (None, 2) с режимом активации SoftMax (рис. 11)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 350)               45150

 dense_1 (Dense)             (None, 50)                17550

 dense_2 (Dense)             (None, 2)                 102

=================================================================
Total params: 62802 (245.32 KB)
Trainable params: 62802 (245.32 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Рис. 11: Слои персептрона

**Dense** – полносвязный слой, где каждый из нейронов НС связан с любым другим из предыдущего слоя. У этого слоя есть такие характеристики как количество нейронов, функция активации, инициализация весов и смещение. [12]

**Количество нейронов** определяет количество выходов, которые слой может сгенерировать и напрямую зависит от размерности входных данных. С его помощью можно управлять сложностью и «гибкостью» извлекаемых признаков.

**Функция активации** задает нелинейность в обработке данных и помогает изучить сложные шаблоны данных. Функция **ReLu** (Rectified Linear Unit) определяется по формуле

$$f(x) = max(0, x)$$

, которая помогает сохранить линейных характер положительных значений и решить проблему исчезающего градиента. Функция SoftMax преобразует вектор входных значений в вектор вероятности с общей суммой всех вероятностей 1, поэтому ее удобно применить в случае классификации на последнем слое.

**Инициализация весов** определяет скорость и качество обучения. Например, если мы стартуем с локального минимума или точки, близкой к нему, то шаг градентного спуска, а следовательно и прирост точности у модели будет минимальным. В стандартном Dense слое по умолчанию стоит **инициализация Глорота**, который устанавливает веса с медианой 0 и дисперсией равной обратной величие к среднему количеству входов и выходов нейрона. Этот метод помогает поддерживать градиенты в управляемом диапазоне. **Инициализация Не** задает веса случайным образом, однако используя при этом нормальное распределение и дисперсию, равную удвоенной величине, обратной к среднему количеству входов в нейрон. При Не учитывается количество нейронов в предыдущем слое, чтобы уменьшить вероятность неконтроллируемых градиентов (уменьшающихся или взрывающихся).

**Смещение** регулирует выход каждого нейрона, добавляя дополнительный параметр. По умолчанию в Dense слое стоит нулевое смещение.

Процесс обучения MLP иллюстрируют графики на рис.12. После 4-5 эпохи прирост точности практически нулевой, это объясняется довольно малым количеством данных для нейронной сети. Итоговый результат у MLP: Loss - 0.12, Accuracy: 96%

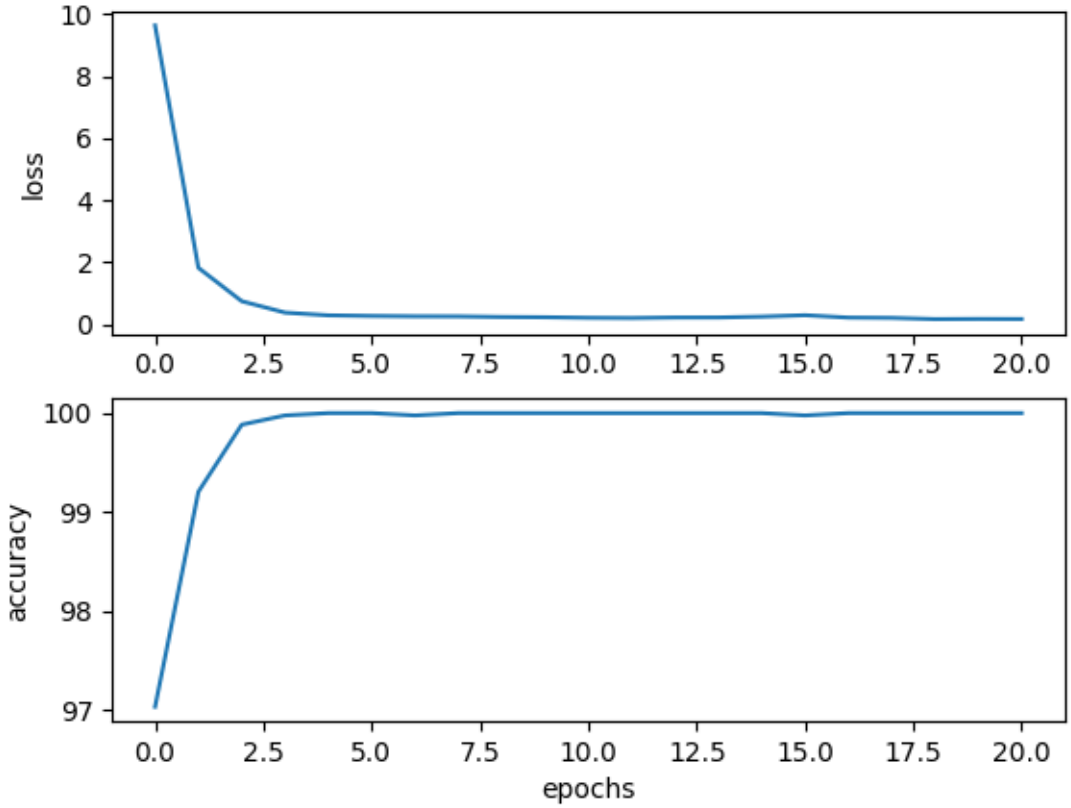


Рис. 12: Графики зависимости потерь и точности MLP от количества эпох

# Заключение

В результате проделанной работы были сгенерированы датасеты гамильтоновости графа, а также код в Maple 2024, способный создать набор данных с любым количеством графов и вершин в них. Все материалы и датасеты опубликованы в открытом доступе на Github, а также лежат на диске. Данный проект имеет перспективы развития в генерации более универсальных датасетов больших размеров на более производительном железе.

Все поставленные в начале задачи были выполнены. Удалось добиться точности в 95.6% при использовании модели многослойного персептрона. Среди моделей машинного обучения лучшую точность показала LinearSVC (95.2%), а также SDG (94.8%) и Gradient Boost (94.7%). Модели Random Forest и Naive Bayes пока-

зали себя хуже - 93.9% и 93.5% соответственно.

В дальнейшей перспективе развития проекта можно опробовать работу с другими методами представления графов в памяти компьютера, такими как матрица смежности или матрица признаков. Работа с графом как с изображением или совокупностью значащих признаков может как послужить для увеличения точности предсказаний, так и для новых эвристических предположений.

Еще одним вектором развития проекта может послужить углубленное математическое исследование свойств гамильтоновости графов и вывод новых закономерностей. Можно провести EDA (Exploriation Data Analysis) зависимостей и корреляций различных характеристик графа и его гамильтоновости. Самый простой из аспектов - количество ребер. Очевидно, что чем больше ребер в графе, тем больше вероятность, что он гамильтонов.

Возможно стоит провести минорные изменения в коде для его оптимизации. Например, одна из ячеек в Collab иногда не работает с первого раза, вне зависимости от обстоятельств.

Во время работы над проектом были углублены знания в сфере машинного обучения и нейронных сетей, проведена оптимизация кода с Python 2.7 на Python 3 и исследование методов Graph2Vec и SkipGram, использующихся и в других областях ML, например в NLP.

# Список литературы


[1] Jianliang Xu Yun Peng, Byron Choi. Graph Learning for Combinatorial Optimization: A Survey of State-of-the-Art / Jianliang Xu Yun Peng, Byron Choi. https://arxiv.org/pdf/2008.12646.

[2] Jean-Claude Bermond. Hamiltonian Graphs / Jean-Claude Bermond // Selected topics in graph theory. — 1979. — no. 6. — Pp. 127–167. https://inria.hal.science/hal-02352666/document.

[3] Ф. Харари. ТЕОРИЯ ГРАФОВ / Ф. Харари. — М.: Мир, 1973. — Р. 300 стр.

[4] Гамильтоновы графы. https://portal.tpu.ru/SHARED/t/TRACEY/Courses/English/Tab1/graph_lec_08.pdf.

[5] FFormula. Способы хранения графа в памяти компьютера / FFormula. https://habr.com/ru/companies/otus/articles/675730/.

[6] Annamalai Narayanan. graph2vec: Learning Distributed Representations of Graphs / Annamalai Narayanan et al. https://arxiv.org/pdf/1707.05005.

[7] Arlei Silva. Machine Learning with Graphs: Representation learning 1/3 - Introduction, Random Walk based Embedding / Arlei Silva. — Spring 2022. https://cs.rice.edu/~al110/teaching/ssj_S22/lecture_3_representation_learning_1.pdf.

[8] Annamalai Narayanan. graph2vec_tf / Annamalai Narayanan. https://github.com/annamalai-nr/graph2vec_tf.git.

[9] Feature scaling in machine learning. https://www.blog.trainindata.com/feature-scaling-in-machine-learning/.

[10] Школа анализа данных. Учебник по машинному обучению / Школа анализа данных. https://education.yandex.ru/handbook/ml.

[11] Scikit Learn Documentation. Neural Network Models Supervised / Scikit Learn Documentation. https://scikit-learn.ru/1-17-neural-network-models-supervised/.

[12] Глубокое погружение в Dense слой: от основ до передовых практик. https://www.yourtodo.ru/posts/glubokoe-pogruzhenie-v-dense-sloj-ot-osnov-do-peredovyih-praktik/.