Введение

Задачи распознавания — это одна из важнейших областей, связанных с обработкой и анализом данных. Они находят применение в различных областях, таких как медицина, банковское дело, компьютерное зрение, робототехника и многих других. Решение этих задач может быть достаточно сложным, и часто требует использования вероятностно-статистических методов.

В настоящее время, вероятностно-статистические методы являются одними из наиболее эффективных и широко используемых методов решения задач распознавания. Они позволяют обрабатывать данные, извлекать из них информацию и принимать решения на основе этой информации.

Существует множество вероятностно-статистических методов, которые используются в задачах распознавания и других областях. Некоторые из них включают в себя:

- Наивный байесовский классификатор - простой, но эффективный метод классификации, основанный на теории вероятностей.
- Метод опорных векторов (SVM) - метод машинного обучения, который используется для классификации и регрессии. SVM строит гиперплоскость или набор гиперплоскостей в многомерном пространстве, которые максимально разделяют элементы разных классов.
- Метод главных компонент (PCA) - метод, используемый для снижения размерности данных путем проекции их на более низкоразмерное пространство. Это помогает устранить шум, улучшить качество данных и ускорить алгоритмы обработки данных.
- Линейная регрессия - метод, используемый для анализа отношения между зависимыми и независимыми переменными. Линейная регрессия может быть использована для прогнозирования значений зависимых переменных на основе значений независимых переменных.

- Кластерный анализ - метод, используемый для разделения групп объектов на основе сходства между ними. Кластерный анализ может быть использован для классификации данных или для поиска скрытых структур в данных.

Это только некоторые из методов, используемых в задачах распознавания и других областях. В зависимости от конкретной задачи может быть использован один или несколько из этих методов, а также другие методы машинного обучения и статистические методы.

Постановка задачи

В рамках данной курсовой работы будут рассмотрены различные вероятностно-статистические методы и их применение в задачах распознавания, таких как классификация изображений, распознавание речи, анализ текстов и другие. Будут рассмотрены основные принципы байесовской статистики, метод опорных векторов, метод главных компонент и другие методы, которые широко применяются в задачах распознавания.

Также будет проведен анализ применимости этих методов на реальных данных с использованием языка программирования Python и библиотек для машинного обучения. Будут использованы различные наборы данных, такие как MNIST для распознавания рукописных цифр, CIFAR-10 для классификации изображений и другие. Анализ результатов позволит оценить эффективность и применимость различных методов в реальных задачах.

Итак, данная работа имеет важное практическое значение и может быть полезна для специалистов в области машинного обучения, анализа данных и других смежных областей, которые занимаются задачами распознавания.

Основные сведения

Вероятностно-статистические методы — это методы и подходы, основанные на теории вероятностей и математической статистике. Они используются для анализа и обработки данных, а также для решения различных задач, таких как классификация, регрессия, кластеризация и другие.

Вероятностно-статистические методы могут использоваться для прогнозирования будущих значений, моделирования зависимостей между переменными, оценки параметров распределений и других статистических характеристик, а также для поиска скрытых закономерностей в данных.

Некоторые из основных вероятностно-статистических методов, используемых в машинном обучении, включают в себя байесовскую статистику, методы регрессии, методы классификации, методы кластеризации и методы снижения размерности данных.

Байесовская статистика — это метод статистического вывода, который основан на теореме Байеса. Он используется для оценки вероятности гипотезы на основе имеющихся данных. Байесовские методы могут использоваться для классификации, регрессии и других задач.

Методы регрессии — это методы, используемые для анализа отношения между зависимыми и независимыми переменными. Они могут быть использованы для прогнозирования значений зависимых переменных на основе значений независимых переменных.

Методы классификации — это методы, используемые для разделения объектов на группы на основе их свойств. Они могут быть использованы для классификации изображений, распознавания речи, анализа текстов и других задач.

Методы кластеризации — это методы, используемые для разделения объектов на группы на основе сходства между ними. Они могут быть использованы для классификации данных или для поиска скрытых структур в данных.

Методы снижения размерности данных — это методы, используемые для уменьшения размерности данных путем проекции их на более низкоразмерное пространство. Это может помочь устранить шум, улучшить качество данных и ускорить алгоритмы обработки данных.

Вероятностно-статистические методы являются важным инструментом в машинном обучении и анализе данных и широко применяются в различных областях, таких как медицина, банковское дело, компьютерное зрение, робототехника и другие. Важно отметить, что выбор конкретного вероятностно-статистического метода зависит от конкретной задачи и особенностей данных, а также от целей и требований заказчика. Корректный выбор метода и оценка его эффективности являются важными задачами при решении задач распознавания и других задач анализа данных.

Многие вероятностно-статистические методы реализованы в библиотеках для машинного обучения, таких как Scikit-learn и TensorFlow. Использование этих библиотек может значительно упростить процесс анализа данных и решения задач распознавания. Однако, важно иметь понимание основных принципов и методов, чтобы правильно интерпретировать результаты и выбрать наиболее подходящий метод для конкретной задачи.

Задачи распознавания — это задачи, связанные с выделением информации из различных источников данных, таких как изображения, звуковые сигналы, тексты и другие данные. Целью задач распознавания является автоматическое определение характеристик или свойств объектов на основе их представления в виде данных.

Примеры задач распознавания включают в себя:

- Распознавание рукописных цифр на изображениях;
- Распознавание лиц на изображениях или видео;
- Распознавание речи и преобразование речи в текст;
- Анализ текстов и определение их темы или настроения;
- Классификация изображений на основе их содержания, например, классификация изображений животных, растений или транспортных средств.

Для решения задач распознавания часто используются методы машинного обучения, такие как нейронные сети, метод опорных векторов, байесовские классификаторы, а также вероятностно-статистические методы и другие. Кроме того, для решения задач распознавания также могут быть использованы методы обработки сигналов и изображений, такие как фильтры и преобразования Фурье.

Наивный байесовский классификатор (НБК)

Наивный байесовский классификатор — это алгоритм машинного обучения, основанный на байесовской статистике и используемый для решения задач классификации. Он основан на предположении о независимости признаков объекта друг от друга.

Алгоритм наивного байесовского классификатора состоит из следующих шагов:

1. Подготовка обучающей выборки;
2. Оценка априорной вероятности каждого класса;
3. Оценка условной вероятности каждого признака для каждого класса;
4. Определение апостериорной вероятности для каждого класса на основе формулы Байеса;
5. Классификация объекта на основе максимальной апостериорной вероятности.

Наивный байесовский классификатор может быть использован для решения задач бинарной и многоклассовой классификации. Для бинарной классификации алгоритм использует один классификатор, который выдаёт вероятность принадлежности объекта к классу 1. Для многоклассовой классификации используется несколько классификаторов, каждый из которых относится к своему классу.

Преимущества наивного байесовского классификатора:

- Простота и скорость алгоритма;
- Хорошая производительность на больших объемах данных;

- Может быть использован для категориальных и непрерывных признаков.

Недостатки наивного байесовского классификатора:

- Ограниченность предположения о независимости признаков;
- Может приводить к неправильным результатам, если условная независимость не выполняется;
- Требует большого количества данных для точных оценок вероятностей.

Это основные принципы и свойства наивного байесовского классификатора. Однако, существует множество вариаций наивного байесовского классификатора, таких как мультиномиальный, бернуллиев, гауссовский и другие. Каждая из этих вариаций имеет свои особенности и может быть использована в различных задачах классификации.

Реализовать наивный байесовский классификатор на Python с использованием библиотеки Scikit-learn для многоклассовой классификации текстовых данных можно следующим образом:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score


def naive_bayesian_classifier(X_train, y_train, X_test, y_test):
    # Создаем векторизатор
    vectorizer = CountVectorizer()
    # Преобразуем текст в числовые признаки
    X_train_vec = vectorizer.fit_transform(X_train)
    # Преобразуем текст тестовой выборки в числовые признаки
    X_test_vec = vectorizer.transform(X_test)
    # Создаем классификатор на основе наивного байесовского алгоритма
    clf = MultinomialNB()
    # Обучаем классификатор на обучающей выборке
```

```
    clf.fit(X_train_vec, y_train)
    # Предсказываем метки классов для тестовой выборки
    y_pred = clf.predict(X_test_vec)

    return (y_pred, accuracy_score(y_test, y_pred))


# Создаем обучающую выборку
X_train = ["This is a good movie",
           "This is a bad movie",
           "This movie is not good",
           "This movie is good",
           "This movie is very bad"]

# Задаем метки классов для обучающей выборки
y_train = ["positive", "negative", "negative", "positive", "negative"]

# Создаем тестовую выборку
X_test = ["This is a very good movie",
          "This is a very bad movie"]

# Вычисляем точность классификации
y_test = ['positive', 'negative']
output = naive_bayesian_classifier(X_train, y_train, X_test, y_test)
print(
    'Ожидаемые данные:',
    y_test,
    '\nРезультаты классификации:',
    output[0],
    '\nТочность классификации:',
    output[1]
)
```

Этот пример демонстрирует, как использовать наивный байесовский классификатор для классификации текстовых данных на два класса (положительные и отрицательные отзывы). Векторизатор CountVectorizer используется для преобразования текста в числовые признаки, а

классификатор MultinomialNB используется для обучения и прогнозирования меток классов. Результаты классификации выводятся на экран, а также вычисляется точность классификации на тестовой выборке.

Результат работы программы:

```
Ожидаемые данные: ['positive', 'negative']
Результаты классификации: ['positive' 'negative']
Точность классификации: 1.0
```

Вывод: точность работы классификатора равна 1 – это означает что все отзывы правильно сгруппированы. Это довольно простой пример, основанный на небольшой выборке исходных данных. При использовании большого числа данных для обучения придется вручную классифицировать отзывы.

Второй пример: классификация текстовых документов на несколько категорий. Исходные данные для классификации текстовых документов были взяты из корпуса текстовых документов "20 Newsgroups", который является стандартным набором данных для задач классификации текстовых документов. Этот корпус содержит текстовые документы из 20 различных категорий новостных групп, таких как компьютерное железо, религия, политика, спорт и т.д.

Корпус "20 Newsgroups" был собран в 1995 году для проведения экспериментов в области обработки естественного языка и классификации текстовых документов. С тех пор он был использован во многих исследованиях и стал стандартным набором данных для задач классификации текстовых документов. В настоящее время этот набор данных доступен для загрузки на многих ресурсах в Интернете.

Код программы:

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report

# Загружаем данные (категории новостей) из библиотеки Scikit-learn
newsgroups_train = fetch_20newsgroups(subset='train')
newsgroups_test = fetch_20newsgroups(subset='test')

# Преобразуем текст в числовые признаки с помощью векторизатора
CountVectorizer
vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(newsgroups_train.data)
X_test = vectorizer.transform(newsgroups_test.data)

# Создаем классификатор на основе наивного байесовского алгоритма
clf = MultinomialNB()

# Обучаем классификатор на обучающей выборке
clf.fit(X_train, newsgroups_train.target)

# Предсказываем метки классов для тестовой выборки
y_pred = clf.predict(X_test)

# Выводим отчет о классификации, включая точность, полноту и F-меру
print(classification_report(
    newsgroups_test.target,
    y_pred,
    target_names=newsgroups_test.target_names
))
```

Этот пример демонстрирует, как использовать наивный байесовский классификатор для классификации текстовых документов на несколько категорий (категории новостей). Векторизатор CountVectorizer используется для преобразования текста в числовые признаки, а классификатор

MultinomialNB используется для обучения и прогнозирования меток классов. Отчет о классификации выводится на экран, включая точность, полноту и F-меру.

Результат работы программы:

	precision	recall	f1-score	support
alt.atheism	0.79	0.77	0.78	319
comp.graphics	0.67	0.74	0.70	389
comp.os.ms-windows.misc	0.20	0.00	0.01	394
comp.sys.ibm.pc.hardware	0.56	0.77	0.65	392
comp.sys.mac.hardware	0.84	0.75	0.79	385
comp.windows.x	0.65	0.84	0.73	395
misc.forsale	0.93	0.65	0.77	390
rec.autos	0.87	0.91	0.89	396
rec.motorcycles	0.96	0.92	0.94	398
rec.sport.baseball	0.96	0.87	0.91	397
rec.sport.hockey	0.93	0.96	0.95	399
sci.crypt	0.67	0.95	0.78	396
sci.electronics	0.79	0.66	0.72	393
sci.med	0.87	0.82	0.85	396
sci.space	0.83	0.89	0.86	394
soc.religion.christian	0.70	0.96	0.81	398
talk.politics.guns	0.69	0.91	0.79	364
talk.politics.mideast	0.85	0.94	0.89	376
talk.politics.misc	0.58	0.63	0.60	310
talk.religion.misc	0.89	0.33	0.49	251
accuracy			0.77	7532
macro avg	0.76	0.76	0.75	7532
weighted avg	0.76	0.77	0.75	7532

Интерпретация выходных данных: эти выходные данные представляют собой отчет о классификации текстовых документов на несколько категорий

с использованием многоклассового классификатора на основе наивного байесовского алгоритма.

Каждая строка отчета соответствует категории, а каждый столбец - это мера (precision, recall, F1-score) для этой категории. В таблице показаны значения мер точности (precision), полноты (recall) и F1-меры (F1-score) для каждой категории.

Например, первая строка отчета: означает, что классификатор правильно классифицировал 79% текстовых документов в категории "alt.atheism" (точность), из всех документов, которые были помечены как "alt.atheism", классификатор правильно идентифицировал 77% (полнота), а F1-мера (среднее гармоническое точности и полноты) равна 0.78.

В конце отчета также показана общая точность классификации (accuracy), которая определяется как отношение числа правильно классифицированных документов ко всем документам. В этом случае, точность классификации равна 0.77, что означает, что 77% всех текстовых документов были правильно классифицированы.

Пример 3 – классификация спам-сообщений. Исходные данные взяты из репозитория на Github по ссылке: https://github.com/rajeevratan84/datascienceforbusiness/blob/master/spam.csv. В этом файле около 5500 строчек данных. Он часто применяется для машинного обучения и анализа данных.

Код программы:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix

# Загружаем данные (сообщения) из CSV-файла
```

```
data = pd.read_csv('spam.csv', encoding='latin-1')

# Разделяем данные на обучающую и тестовую выборки
X_train = data.v2[:4400].values
X_test = data.v2[4400:].values
y_train = data.v1[:4400].values
y_test = data.v1[4400:].values

# Преобразуем текст в числовые признаки с помощью векторизатора CountVectorizer
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Создаем классификатор на основе наивного байесовского алгоритма
clf = MultinomialNB()

# Обучаем классификатор на обучающей выборке
clf.fit(X_train_vec, y_train)

# Предсказываем метки классов для тестовой выборки
y_pred = clf.predict(X_test_vec)

# Выводим матрицу ошибок и точность классификации
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Этот пример демонстрирует, как использовать наивный байесовский классификатор для классификации спам-сообщений. Данные (сообщения) загружаются из CSV-файла и разделяются на обучающую и тестовую выборки. Векторизатор CountVectorizer используется для преобразования текста в числовые признаки, а классификатор MultinomialNB используется для обучения и прогнозирования меток классов. Матрица ошибок и точность классификации выводятся на экран.

Результат работы программы:

```
Confusion Matrix:
 [[1015    8]
 [   8  141]]
Accuracy: 0.9863481228668942
```

Интерпретация выходных данных: матрица ошибок (Confusion matrix) — это таблица, которая показывает число верно и неверно классифицированных примеров в каждой из двух категорий. В данном случае, первая строка и первый столбец матрицы соответствуют категории 0 (не спам), а вторая строка и второй столбец - категории 1 (спам). Таким образом, матрица ошибок имеет следующий вид:

- [[1015 8] – 1015 не-спам сообщений правильно классифицированы, 8 неправильно классифицированы как спам
- [8 141]] – 141 спам сообщений правильно классифицированы, 8 неправильно классифицированы как не-спам

Точность (Accuracy) — это мера правильности классификации, которая определяется как отношение числа верно классифицированных примеров ко всем примерам. В данном случае, точность равна 0.9863, что означает, что 98,63% всех сообщений были правильно классифицированы.

Подведем итоги: "Наивный байесовский классификатор" - это простой и быстрый алгоритм машинного обучения для классификации текстовых документов на основе вероятностной модели. Алгоритм основан на предположении о независимости факторов, что делает его "наивным".

Основные преимущества наивного байесовского классификатора:

- Простота: алгоритм быстро реализуется и обучается.
- Низкие вычислительные затраты: алгоритм работает быстро и требует относительно мало ресурсов.

- Хорошая производительность: наивный байесовский классификатор может давать хорошие результаты в задачах классификации текстовых документов.
- Хорошая интерпретируемость: алгоритм основан на вероятностной модели, что делает его результаты легко интерпретируемыми.

Некоторые из недостатков наивного байесовского классификатора:

- Предположение о независимости факторов может быть неверным для некоторых наборов данных.
- Алгоритм может давать неоптимальные результаты в задачах, в которых существует много зависимых переменных.
- Некоторые типы данных могут не подходить для использования с наивным байесовским классификатором.

Несмотря на эти ограничения, наивный байесовский классификатор по-прежнему широко используется в задачах классификации текстовых документов и считается одним из самых эффективных алгоритмов для этой задачи.

Метод опорных векторов (SVM)

Метод опорных векторов (SVM) может быть применен для различных задач классификации и регрессии, включая распознавание образов, классификацию текстовых данных, биоинформатику, финансовые прогнозы, анализ изображений и многие другие.

SVM может быть особенно полезен в задачах классификации, когда данных много и они линейно разделимы. В таких случаях SVM может дать лучшие результаты, чем наивный байесовский классификатор. Кроме того, SVM может быть использован для работы с нелинейно разделимыми данными, благодаря использованию функций ядра.

Однако, для малых объемов данных, наивный байесовский классификатор может быть более эффективным и простым в использовании, чем SVM. Наивный байесовский классификатор также может дать хорошие результаты для задач классификации текстовых данных и других типов данных, когда данные имеют простую структуру и нет сильной зависимости между признаками.

Таким образом, выбор между SVM и наивным байесовским классификатором зависит от типа задачи и объема данных, которые необходимо обработать. Если данных много, и они сложные, SVM может быть более эффективным. Если же данных мало, и они простые, то наивный байесовский классификатор может быть более подходящим выбором.

Сравним результаты классификации новостных статей, реализованной с помощью метода опорных векторов и наивного байесовского классификатора.

Код программы:

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report

# Загрузка данных из корпуса "20 Newsgroups"
newsgroups = fetch_20newsgroups(subset='all')

# Преобразование текстовых данных в числовой формат
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(newsgroups.data)

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X,
newsgroups.target, test_size=0.3, random_state=42)

# Обучение SVM
clf = SVC(kernel='linear', C=1, gamma='auto')
clf.fit(X_train, y_train)

# Оценка качества модели на тестовой выборке
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred,
target_names=newsgroups.target_names))
```

Результаты:

```
                          precision    recall  f1-score   support

             alt.atheism       0.94      0.92      0.93       236
           comp.graphics       0.74      0.86      0.79       287
 comp.os.ms-windows.misc       0.86      0.84      0.85       290
comp.sys.ibm.pc.hardware       0.76      0.80      0.78       285
   comp.sys.mac.hardware       0.91      0.86      0.89       312
```

```
                comp.windows.x       0.89      0.88      0.88       308
                  misc.forsale       0.81      0.87      0.84       276
                     rec.autos       0.93      0.93      0.93       304
               rec.motorcycles       1.00      0.95      0.97       279
            rec.sport.baseball       0.98      0.97      0.97       308
              rec.sport.hockey       0.97      0.96      0.97       309
                     sci.crypt       0.99      0.96      0.97       290
               sci.electronics       0.85      0.86      0.86       304
                       sci.med       0.96      0.93      0.95       300
                     sci.space       0.96      0.95      0.95       297
    soc.religion.christian       0.95      0.98      0.96       292
        talk.politics.guns       0.95      0.96      0.95       270
     talk.politics.mideast       1.00      0.98      0.99       272
        talk.politics.misc       0.95      0.92      0.93       239
        talk.religion.misc       0.92      0.84      0.88       196

                  accuracy                           0.91      5654
                 macro avg       0.91      0.91      0.91      5654
              weighted avg       0.91      0.91      0.91      5654
```

Выводы: метод опорных векторов справился с задачей классификации статей гораздо лучше, чем НБК. Однако, замечу, на обучение модели потребовалось больше времени – в 5 раз больше.

Пример 2 – распознавание текста. Для распознавания текста необходимо обучить модель на большом числе исходных данных. Для этого можно воспользоваться встроенными в библиотеку sklearn массивом исходных данных. Алгоритм выглядит аналогично примеру с классификацией новостных статей. Поэтому покажу только результат обучения:

```
Accuracy: 0.9796296296296296
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00        53
           1       0.98      0.98      0.98        50
           2       0.98      1.00      0.99        47
           3       1.00      0.96      0.98        54
           4       0.98      0.98      0.98        60
           5       0.97      0.97      0.97        66
           6       1.00      1.00      1.00        53
           7       0.96      0.98      0.97        55
           8       0.95      0.98      0.97        43
           9       0.97      0.95      0.96        59

    accuracy                           0.98       540
   macro avg       0.98      0.98      0.98       540
weighted avg       0.98      0.98      0.98       540
```

Чтобы применить обученную модель на живом примере и распознать число на картинке, нам нужно выполнить следующие шаги:

1. Загрузить изображение с числом, которое мы хотим распознать.
2. Обработать изображение, чтобы привести его к формату, который может быть использован в качестве входных данных для модели. Обычно это означает изменение размера изображения и преобразование его в черно-белый формат.
3. Подать обработанное изображение на вход обученной модели и получить предсказание класса числа.
4. Отобразить предсказанное число на экране.

Вот пример, который демонстрирует, как выполнить эти шаги с помощью Python и OpenCV:

```
import cv2
import numpy as np
import joblib
```

```
# Загрузка обученной модели
model = joblib.load('svm_model.pkl')

# Загрузка изображения с числом
image = cv2.imread('number.png')

# Преобразование изображения в черно-белый формат
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Изменение размера изображения
resized = cv2.resize(gray, (8, 8), interpolation=cv2.INTER_AREA)

# Преобразование изображения в одномерный массив
flattened = np.ravel(resized)

# Подача массива на вход модели и получение предсказания класса числа
prediction = model.predict([flattened])

# Отображение предсказанного числа на экране
cv2.putText(image, str(prediction[0]), (10, 50),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 2)
cv2.imshow('Number', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Результат показан на рисунке 1

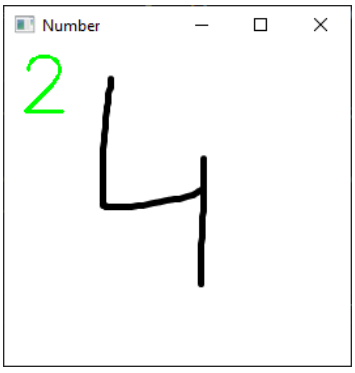

Рисунок 1 – Распознавание рукописного числа на после обучения модели.

Видим, что распознанный результат не совпадает с ожидаемым. Причины могут быть следующими:

- Малый объём обучающих данных;
- Параметры обучаемой модели.
- Переобучение модели

Причем эти причины не исключают друг друга.