объясни код class CountVectorizer:

def \_\_init\_\_(self, max\_df=1.0, min\_df=1):

self.vocab = {}

self.max\_df = max\_df

self.min\_df = min\_df

def fit(self, text):

doc\_freq = {}

for doc in text:

for word in doc.split():

if word not in doc\_freq:

doc\_freq[word] = 0

doc\_freq[word] += 1

if ((type(self.max\_df) is float and doc\_freq[word] / len(text) <= self.max\_df) or doc\_freq[word] <= self.max\_df) and ((type(self.min\_df) and float or doc\_freq[word] / len(text) >= self.min\_df) or doc\_freq[word] >= self.min\_df):

if word not in self.vocab:

self.vocab[word] = len(self.vocab)

def transform(self, text):

X = sparse.csc\_array(np.zeros((len(text), len(self.vocab))))

for i, doc in enumerate(text):

for word in doc.split():

if word in self.vocab:

X[i, self.vocab[word]] += 1

return X

def fit\_transform(self, text):

self.fit(text)

return self.transform(text)

CountVectorizer - это класс, который используется для преобразования текста в числовые векторы. Он имеет следующие методы:

* \_\_init\_\_(self, max\_df=1.0, min\_df=1) - конструктор класса, который принимает два аргумента: max\_df и min\_df. Эти аргументы используются для фильтрации слов, которые встречаются слишком часто или слишком редко в тексте.
* fit(self, text) - метод, который обучает модель на тексте. Он вычисляет частоту каждого слова в тексте и сохраняет слова, которые удовлетворяют условиям max\_df и min\_df, в словаре self.vocab.
* transform(self, text) - метод, который преобразует текст в числовой вектор. Он создает разреженную матрицу X, где каждая строка соответствует документу из текста, а каждый столбец соответствует слову из словаря self.vocab. Затем он заполняет матрицу значениями, равными количеству вхождений каждого слова в каждый документ.
* fit\_transform(self, text) - метод, который сочетает в себе вызовы методов fit и transform. Он обучает модель на тексте и затем преобразует текст в числовой вектор.

2из30

объясни код data = pd.read\_csv("../data/spam.csv", sep=',')["v2"]

data

Этот код использует библиотеку pandas для чтения файла CSV с именем spam.csv, который находится в папке data на уровень выше текущей директории. Файл читается с использованием запятой в качестве разделителя между значениями.

Затем код выбирает столбец с именем v2 из прочитанного файла и сохраняет его в переменной data. После этого он выводит содержимое переменной data на экран.

Таким образом, этот код читает столбец v2 из файла CSV и выводит его содержимое.

3из30

объясни код vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(data)

print(vectorizer.vocab)

print(X)

Этот код создает объект CountVectorizer и сохраняет его в переменной vectorizer. Затем он вызывает метод fit\_transform у этого объекта, передавая ему переменную data в качестве аргумента. Этот метод обучает модель на тексте из переменной data и затем преобразует его в числовой вектор, который сохраняется в переменной X.

Затем код выводит содержимое словаря vectorizer.vocab, который содержит слова, которые удовлетворяют условиям max\_df и min\_df, и их индексы в матрице X. Он также выводит содержимое матрицы X, которая представляет собой разреженную матрицу, где каждая строка соответствует документу из текста, а каждый столбец соответствует слову из словаря vectorizer.vocab.

Таким образом, этот код обучает модель CountVectorizer на тексте из переменной data, преобразует его в числовой вектор и выводит информацию о словаре и матрице.

4из30

объясни код class TfidfVectorizer:

def \_\_init\_\_(self):

self.vocab = {}

self.idf = {}

self.doc\_count = 0

def fit(self, X : list):

self.doc\_count = len(X)

for doc in X:

for word in doc.split():

if word not in self.vocab:

self.vocab[word] = 1

else:

self.vocab[word] += 1

for word in self.vocab:

self.idf[word] = np.log(self.doc\_count / (1 + self.vocab[word]))

def transform(self, X):

tfidf\_matrix = []

for doc in X:

tfidf\_vector = []

for word in self.vocab:

tf = doc.split().count(word)/self.vocab[word]

tfidf = tf \* self.idf[word]

tfidf\_vector.append(tfidf)

tfidf\_matrix.append(tfidf\_vector)

return sparse.csc\_array(np.array(tfidf\_matrix))

def fit\_transform(self, X):

self.fit(X)

return self.transform(X)

TfidfVectorizer - это класс, который используется для преобразования текста в числовые векторы с использованием метрики TF-IDF. Он имеет следующие методы:

* \_\_init\_\_(self) - конструктор класса, который инициализирует словарь self.vocab, словарь self.idf и переменную self.doc\_count.
* fit(self, X : list) - метод, который обучает модель на тексте из списка X. Он вычисляет количество документов в списке и сохраняет его в переменной self.doc\_count. Затем он вычисляет частоту каждого слова в тексте и сохраняет ее в словаре self.vocab. Наконец, он вычисляет обратную документную частоту (IDF) для каждого слова и сохраняет ее в словаре self.idf.
* transform(self, X) - метод, который преобразует текст из списка X в числовой вектор с использованием метрики TF-IDF. Он создает матрицу tfidf\_matrix, где каждая строка соответствует документу из списка X, а каждый столбец соответствует слову из словаря self.vocab. Затем он вычисляет значение TF-IDF для каждого слова в каждом документе и заполняет матрицу этими значениями. В конце он преобразует матрицу в разреженную матрицу и возвращает ее.
* fit\_transform(self, X) - метод, который сочетает в себе вызовы методов fit и transform. Он обучает модель на тексте из списка X и затем преобразует его в числовой вектор с использованием метрики TF-IDF.

5из30

объясни код vectorizer = TfidfVectorizer()

X = vectorizer.fit\_transform(data)

print(vectorizer.vocab)

print(X)

Этот код создает объект TfidfVectorizer и сохраняет его в переменной vectorizer. Затем он вызывает метод fit\_transform у этого объекта, передавая ему переменную data в качестве аргумента. Этот метод обучает модель на тексте из переменной data и затем преобразует его в числовой вектор с использованием метрики TF-IDF, который сохраняется в переменной X.

Затем код выводит содержимое словаря vectorizer.vocab, который содержит слова и их частоты в тексте. Он также выводит содержимое матрицы X, которая представляет собой разреженную матрицу, где каждая строка соответствует документу из текста, а каждый столбец соответствует слову из словаря vectorizer.vocab.

Таким образом, этот код обучает модель TfidfVectorizer на тексте из переменной data, преобразует его в числовой вектор с использованием метрики TF-IDF и выводит информацию о словаре и матрице.

6из30

объясни код vectorizer = TfidfVectorizer()

data = pd.read\_csv("../data/spam.csv", sep=',')

X, y = vectorizer.fit\_transform(data["v2"]), data["v1"]

vectorizer.vocab

Этот код создает объект TfidfVectorizer и сохраняет его в переменной vectorizer. Затем он использует библиотеку pandas для чтения файла CSV с именем spam.csv, который находится в папке data на уровень выше текущей директории. Файл читается с использованием запятой в качестве разделителя между значениями и сохраняется в переменной data.

Затем код вызывает метод fit\_transform у объекта vectorizer, передавая ему столбец с именем v2 из прочитанного файла в качестве аргумента. Этот метод обучает модель на тексте из этого столбца и затем преобразует его в числовой вектор с использованием метрики TF-IDF, который сохраняется в переменной X. Код также сохраняет столбец с именем v1 из прочитанного файла в переменной y.

Наконец, код выводит содержимое словаря vectorizer.vocab, который содержит слова и их частоты в тексте.

Таким образом, этот код обучает модель TfidfVectorizer на тексте из столбца v2 файла CSV, преобразует его в числовой вектор с использованием метрики TF-IDF и выводит информацию о словаре.

7из30

объясни код class PCA\_new():

def \_\_init\_\_(self, n\_components=2):

self.n\_components = n\_components

self.components = None

self.mean = None

def fit(self, X):

self.mean = np.mean(X, axis=0)

X = X - self.mean

cov\_matrix = np.cov(X.T)

eigenvalues, eigenvectors = np.linalg.eig(cov\_matrix)

eigenvectors = eigenvectors.T

idxs = np.argsort(eigenvalues)[::-1]

eigenvalues = eigenvalues[idxs]

eigenvectors = eigenvectors[idxs]

self.components = eigenvectors[0:self.n\_components]

return self

def transform(self, X):

X = X - self.mean

return np.dot(X, self.components.T)

def fit\_transform(self, X):

self.fit(X)

return self.transform(X)

PCA\_new - это класс, который реализует метод главных компонент (PCA) для снижения размерности данных. Он имеет следующие методы:

* \_\_init\_\_(self, n\_components=2) - конструктор класса, который принимает один аргумент: n\_components, который определяет количество главных компонент, которые будут сохранены после снижения размерности. Он также инициализирует переменные self.components и self.mean, которые будут использоваться позже.
* fit(self, X) - метод, который обучает модель на данных из матрицы X. Он вычисляет среднее значение каждого признака и сохраняет его в переменной self.mean. Затем он центрирует данные, вычитая среднее значение из каждого признака. Далее он вычисляет ковариационную матрицу и находит ее собственные значения и собственные векторы. Затем он сортирует собственные значения в порядке убывания и выбирает соответствующие собственные векторы. Наконец, он сохраняет первые n\_components собственных векторов в переменной self.components.
* transform(self, X) - метод, который преобразует данные из матрицы X в новое пространство меньшей размерности. Он центрирует данные, вычитая среднее значение из каждого признака. Затем он умножает центрированные данные на транспонированную матрицу главных компонент, чтобы получить преобразованные данные.
* fit\_transform(self, X) - метод, который сочетает в себе вызовы методов fit и transform. Он обучает модель на данных из матрицы X и затем преобразует их в новое пространство меньшей размерности.

Таким образом, этот класс реализует метод главных компонент для снижения размерности данных.

8из30

объясни код pca = PCA\_new(n\_components=2)

downfeatured = pca.fit\_transform(X).T

colors = data['v1'].replace({'ham': 0, 'spam': 1})

colors

Этот код создает объект PCA\_new с параметром n\_components=2 и сохраняет его в переменной pca. Затем он вызывает метод fit\_transform у этого объекта, передавая ему матрицу X в качестве аргумента. Этот метод обучает модель на данных из матрицы X и затем преобразует их в новое пространство меньшей размерности с двумя главными компонентами. Преобразованные данные транспонируются и сохраняются в переменной downfeatured.

Затем код заменяет значения в столбце с именем v1 из датафрейма data на числовые значения: 0 для 'ham' и 1 для 'spam'. Замененные значения сохраняются в переменной colors.

Таким образом, этот код обучает модель PCA на данных из матрицы X, преобразует их в новое пространство меньшей размерности с двумя главными компонентами и заменяет значения в столбце датафрейма на числовые значения.

9из30

объясни код plt.scatter(x=downfeatured[0], y=downfeatured[1], c=colors);

Этот код использует библиотеку matplotlib для создания диаграммы рассеяния. Он вызывает функцию scatter и передает ей координаты точек по осям x и y, которые соответствуют первой и второй строкам матрицы downfeatured. Он также передает аргумент c, который определяет цвет каждой точки в соответствии с значениями из переменной colors.

Таким образом, этот код создает диаграмму рассеяния, где каждая точка соответствует строке из матрицы downfeatured, а ее цвет определяется значением из переменной colors.

10из30

объясни код X\_train, X\_test, y\_train, y\_test = train\_test\_split(downfeatured.T, colors, test\_size=0.2, random\_state=42)

Этот код использует функцию train\_test\_split из библиотеки scikit-learn для разделения данных на обучающую и тестовую выборки. Он передает функции транспонированную матрицу downfeatured, переменную colors, размер тестовой выборки test\_size=0.2 и начальное значение генератора случайных чисел random\_state=42.

Функция train\_test\_split перемешивает данные и разделяет их на обучающую и тестовую выборки в соответствии с указанным размером тестовой выборки. В этом случае 20% данных будут использоваться в качестве тестовой выборки, а оставшиеся 80% - в качестве обучающей.

Результаты разделения данных сохраняются в переменных X\_train, X\_test, y\_train и y\_test. Переменные X\_train и X\_test содержат признаки для обучающей и тестовой выборок соответственно, а переменные y\_train и y\_test содержат метки классов для обучающей и тестовой выборок соответственно.

Таким образом, этот код разделяет данные на обучающую и тестовую выборки с использованием функции train\_test\_split.

11из30

объясни код class k\_nn:

def \_\_init\_\_(self, X\_train, y\_train):

self.xtr = X\_train

self.ytr = y\_train

def predict(self, X\_test, k):

pred = np.array([])

X\_test = X\_test

for i in range(X\_test.shape[0]):

distances = pd.Series(pow(pow(self.xtr - X\_test[i], 2).sum(axis = 1), 1/2)).sort\_values()[:k]

values = []

for j in distances.index:

values.append(self.ytr[j])

pred = np.append(pred, bool(np.bincount(np.array(values)).argmax()))

return pred

k\_nn - это класс, который реализует алгоритм k-ближайших соседей (k-NN) для классификации данных. Он имеет следующие методы:

* \_\_init\_\_(self, X\_train, y\_train) - конструктор класса, который принимает два аргумента: X\_train и y\_train, которые представляют собой обучающую выборку и соответствующие ей метки классов. Он сохраняет эти данные в переменных self.xtr и self.ytr.
* predict(self, X\_test, k) - метод, который предсказывает метки классов для тестовой выборки X\_test с использованием алгоритма k-NN. Он принимает два аргумента: X\_test и k, где k - это количество ближайших соседей, которые будут использоваться для предсказания меток классов. Метод вычисляет расстояния от каждого объекта тестовой выборки до каждого объекта обучающей выборки и выбирает k ближайших соседей. Затем он определяет наиболее часто встречающийся класс среди этих соседей и сохраняет его в массиве предсказаний. В конце он возвращает массив предсказаний.

Таким образом, этот класс реализует алгоритм k-NN для классификации данных.

12из30

объясни код knn = k\_nn(X\_train, colors)

y\_pred = knn.predict(X\_test, 1)

Этот код создает объект k\_nn и передает ему обучающую выборку X\_train и соответствующие ей метки классов colors. Объект сохраняется в переменной knn. Затем он вызывает метод predict у этого объекта, передавая ему тестовую выборку X\_test и значение k=1 в качестве аргументов. Этот метод предсказывает метки классов для тестовой выборки с использованием алгоритма k-NN с k=1. Предсказанные метки классов сохраняются в переменной y\_pred.

Таким образом, этот код обучает модель k-NN на обучающей выборке и предсказывает метки классов для тестовой выборки с использованием алгоритма k-NN с k=1.

13из30

объясни код class Metrics:

def \_\_init\_\_(self, test, pred):

self.TP = 0

for i, j in zip(test, pred):

if i == j == True:

self.TP += 1

self.FN = np.count\_nonzero(test == True) - self.TP

self.FP = np.count\_nonzero(pred == True) - self.TP

self.TN = test.size - self.TP - self.FP - self.FN

self.P = self.TP + self.FN

self.N = self.TN + self.FP

def conf\_matr(self):

print('%-7s %-7s' % ('ㅤ', 'реальность'))

print('%-7s %5d %5d' % ('прог', self.TP, self.FP))

print('%-7s %5d %5d' % ('ноз', self.FN, self.TN))

def accuracy(self):

return (self.TP + self.TN)/(self.P + self.N)

def precision(self):

return self.TP/(self.TP + self.FP)

def recall(self):

return self.TP/self.P

def F1(self):

return 2/(1/self.precision() + 1/self.recall())

Metrics - это класс, который используется для вычисления различных метрик качества классификации. Он имеет следующие методы:

* \_\_init\_\_(self, test, pred) - конструктор класса, который принимает два аргумента: test и pred, которые представляют собой истинные метки классов и предсказанные метки классов соответственно. Он вычисляет значения истинно положительных (TP), ложно отрицательных (FN), ложно положительных (FP) и истинно отрицательных (TN) предсказаний, а также общее количество положительных (P) и отрицательных (N) объектов.
* conf\_matr(self) - метод, который выводит на экран матрицу ошибок (confusion matrix), которая показывает количество TP, FP, FN и TN предсказаний.
* accuracy(self) - метод, который вычисляет точность (accuracy) классификации как долю правильных предсказаний среди всех объектов.
* precision(self) - метод, который вычисляет точность (precision) классификации как долю истинно положительных предсказаний среди всех положительных предсказаний.
* recall(self) - метод, который вычисляет полноту (recall) классификации как долю истинно положительных предсказаний среди всех положительных объектов.
* F1(self) - метод, который вычисляет F1-меру как гармоническое среднее между точностью и полнотой.

Таким образом, этот класс используется для вычисления различных метрик качества классификации.

14из30

объясни код knn\_m = Metrics(y\_test, y\_pred)

knn\_m.conf\_matr()

print(knn\_m.accuracy())

print(knn\_m.precision())

print(knn\_m.recall())

print(knn\_m.F1())

Этот код создает объект Metrics и передает ему истинные метки классов y\_test и предсказанные метки классов y\_pred. Объект сохраняется в переменной knn\_m. Затем он вызывает метод conf\_matr у этого объекта, который выводит на экран матрицу ошибок.

Затем код вызывает методы accuracy, precision, recall и F1 у объекта knn\_m, которые вычисляют соответствующие метрики качества классификации и выводят их на экран.

Таким образом, этот код вычисляет различные метрики качества классификации для предсказанных меток классов и выводит их на экран.

15из30

объясни код from sklearn.decomposition import LatentDirichletAllocation

from sklearn.feature\_extraction.text import CountVectorizer

Этот код импортирует два класса из библиотеки scikit-learn: LatentDirichletAllocation и CountVectorizer.

LatentDirichletAllocation - это класс, который реализует алгоритм скрытого размещения Дирихле (LDA) для тематического моделирования текстовых данных. Он используется для выявления скрытых тем в коллекции текстовых документов и преобразования их в числовые векторы.

CountVectorizer - это класс, который используется для преобразования текста в числовые векторы с использованием подсчета слов. Он вычисляет частоту каждого слова в тексте и преобразует текст в числовой вектор, где каждый элемент соответствует количеству вхождений определенного слова в текст.

Таким образом, этот код импортирует два класса из библиотеки scikit-learn, которые используются для тематического моделирования и преобразования текста в числовые векторы.

16из30

объясни код data = pd.read\_csv("../data/spam.csv", sep=',')['v2']

Этот код использует библиотеку pandas для чтения файла CSV с именем spam.csv, который находится в папке data на уровень выше текущей директории. Файл читается с использованием запятой в качестве разделителя между значениями.

Затем код выбирает столбец с именем v2 из прочитанного файла и сохраняет его в переменной data.

Таким образом, этот код читает столбец v2 из файла CSV и сохраняет его содержимое в переменной data.

17из30

объясни код tf\_vectorizer = CountVectorizer(max\_df=0.9, min\_df=5,

max\_features=1000,

stop\_words='english')

tf = tf\_vectorizer.fit\_transform(data)

Этот код создает объект CountVectorizer с указанными параметрами и сохраняет его в переменной tf\_vectorizer. Параметры max\_df=0.9 и min\_df=5 определяют максимальную и минимальную документную частоту слов, которые будут включены в словарь. Параметр max\_features=1000 определяет максимальное количество слов, которые будут включены в словарь. Параметр stop\_words='english' указывает, что нужно использовать список английских стоп-слов для удаления нерелевантных слов из текста.

Затем код вызывает метод fit\_transform у объекта tf\_vectorizer, передавая ему переменную data в качестве аргумента. Этот метод обучает модель на тексте из переменной data и затем преобразует его в числовой вектор с использованием подсчета слов. Преобразованный текст сохраняется в переменной tf.

Таким образом, этот код создает объект CountVectorizer с указанными параметрами, обучает его на тексте из переменной data и преобразует текст в числовой вектор с использованием подсчета слов.

18из30

объясни код lda = LatentDirichletAllocation(n\_components=10, max\_iter=20,

learning\_method='online',

learning\_offset=50.,

random\_state=0).fit(tf)

Этот код создает объект LatentDirichletAllocation с указанными параметрами и сохраняет его в переменной lda. Параметр n\_components=10 определяет количество тем, которые будут выявлены в тексте. Параметр max\_iter=20 определяет максимальное количество итераций алгоритма. Параметр learning\_method='online' указывает, что нужно использовать онлайн-метод обучения. Параметр learning\_offset=50. определяет смещение для онлайн-метода обучения. Параметр random\_state=0 определяет начальное значение генератора случайных чисел.

Затем код вызывает метод fit у объекта lda, передавая ему переменную tf в качестве аргумента. Этот метод обучает модель LDA на тексте из переменной tf.

Таким образом, этот код создает объект LatentDirichletAllocation с указанными параметрами и обучает его на тексте из переменной tf.

19из30

объясни код for topic\_idx, topic in enumerate(lda.components\_):

print(f"Topic {topic\_idx}")

print(" ".join([tf\_vectorizer.get\_feature\_names\_out()[i]

for i in topic.argsort()[:-11:-1]]))

Этот код использует цикл for для перебора тем, выявленных моделью LDA, которая сохранена в переменной lda. Он перебирает элементы массива lda.components\_, который содержит вероятности слов для каждой темы, и сохраняет индекс темы в переменной topic\_idx, а вероятности слов - в переменной topic.

Затем код выводит на экран номер темы и 10 наиболее вероятных слов для этой темы. Он сортирует вероятности слов в порядке убывания и выбирает 10 наибольших. Затем он использует метод get\_feature\_names\_out у объекта tf\_vectorizer для получения списка всех слов и выбирает соответствующие слова из этого списка. Наконец, он объединяет выбранные слова в строку с разделителем-пробелом и выводит ее на экран.

Таким образом, этот код выводит на экран 10 наиболее вероятных слов для каждой темы, выявленной моделью LDA.