Классификация тональности текстов с использованием методов машинного обучения

Гольдберг Артемий

Осень, 2025

Постановка задачи

Цель проекта — построить модель анализа тональности коротких текстов (отзывов):

- вход: текст на английском языке;
- выход: метка **позитивный** / **негативный**;
- обучаем и сравниваем несколько классических моделей;
- интегрируем обученные модели в веб-приложение `Streamlit`.

Датасеты

Используются два открытых англоязычных набора:

- **Kaggle `train.csv`** (классический датасет по отзывам)
  - размеченные отзывы с колонками `Sentiment` и `Text`;
  - `Sentiment` $\in \{0, 1\}$: 0 — негатив, 1 — позитив;
  - несколько десятков тысяч сообщений.
- **Sentiment140 (`training.noemoticon.csv`)**
  - около $1{,}6$ млн отзывов;
  - исходные метки 0 / 4 приводим к 0 / 1;
  - вариант `noemoticon`: без эмодзи.
- **Итоговая структура:**

$$(\text{text},\ \text{label}), \qquad \text{label} \in \{0, 1\}.$$

Предобработка текста

NLP-pipeline перед векторизацией:

- перевод в нижний регистр;
- токенизация по регулярному выражению;
- удаление английских стоп-слов (NLTK `stopwords`);
- стемминг (PorterStemmer);
- сборка очищенного текста обратно в строку.

Цель — уменьшить размер словаря и убрать шум.

Примеры до / после предобработки

Позитивный пример

До:

This movie was really good, I enjoyed every minute of it!

После:

`movi realli good enjoy everi minut`

Негативный пример

До:

This movie was terrible and boring, I would not recommend it.

После:

`movi terribl bore would not recommend`

Комментарий: форма слова не важна, важен сам «корень» и его вес в документе.

TF-IDF

TF-IDF оценивает «важность» слова в документе:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \cdot \text{IDF}(t)$$

$$\text{TF}(t, d) = \frac{\text{число вхождений } t \text{ в } d}{\text{общее число слов в } d}$$

$$\text{IDF}(t) = \log \frac{N}{1 + df(t)}$$

- частые во всех документах слова (*the, and, of*) получают малый вес;
- редкие, но характерные слова — больший вес;
- на выходе: разреженный вектор признаков для каждого текста.

Реализация проекта

1. загрузка и объединение датасетов;
2. предобработка текста (NLP-pipeline);
3. разбиение на train / test;
4. построение TF-IDF признаков;
5. обучение трёх моделей:
   - Logistic Regression;
   - Linear SVM (Calibrated);
   - Multinomial Naive Bayes;
6. оценка качества и выбор лучшей модели;
7. экспорт TF-IDF и моделей в `joblib`;
8. интеграция в веб-приложение Streamlit.

Модель 1: Logistic Regression

Логистическая регрессия — линейный классификатор:

$$P(y = 1 \mid x) = \sigma(w^T x + b), \qquad \sigma(z) = \frac{1}{1 + e^{-z}}.$$

Оптимизация:

$$\min_{w,b} \left[-\sum_i y_i \log p_i + (1 - y_i) \log(1 - p_i) \right] + \lambda \|w\|^2.$$

Плюсы:

- интерпретируемые веса слов;
- хорошо работает с TF-IDF;
- даёт вероятности классов.

Минус: линейная граница, слабо ловит сложные нелинейности.

Модель 2: Linear SVM (Calibrated)

SVM ищет гиперплоскость максимального зазора:

$$w^T x + b = 0,$$

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_i \xi_i.$$

- используем `LinearSVC` + `CalibratedClassifierCV` (метод `"sigmoid"`) для получения вероятностей;
- хорошо работает на больших разреженных матрицах признаков.

Плюсы:

- устойчива к переобучению;
- высокая точность на текстовых задачах.

Минусы:

- изначально не даёт вероятностей;
- калибровка — дополнительный шаг обучения.

Модель 3: Multinomial Naive Bayes

Байесовский подход к частотам слов:

$$P(y \mid x) \propto P(y) \prod_i P(x_i \mid y),$$

где x_i — количество вхождений слова i.
Предположение: признаки (слова) условно независимы по классу.

Плюсы:

- очень простая и быстрая модель;
- классический базовый метод для текстов.

Минусы:

- независимость слов в реальных текстах не выполняется;
- как правило уступает LogReg / SVM по качеству.

Как обучали модели

1. **Разделение данных**
   - объединённый корпус → train / test;
   - стратифицированное разбиение по меткам.
2. **Обучение векторизатора**
   - `TfidfVectorizer` на train-части;
   - диапазон `ngram_range=(1,5)`;
   - игнорирование слишком редких / слишком частых слов.
3. **Подбор гиперпараметров**
   - `GridSearchCV` по параметрам C, `class_weight` (LogReg, SVM);
   - выбор лучшей модели по метрике F1 (взвешенная).
4. **Финальное обучение**
   - дообучение лучших моделей на всём train;
   - оценка на отложенной test-выборке.

Время обучения моделей

Обучение моделей выполнялось оффлайн в Jupyter-ноутбуке, один раз, на объединённом корпусе твитов после TF–IDF-векторизации.

Основные факторы времени обучения:

- размер корпуса (число отзывов и длина текстов);
- размер словаря TF–IDF (n-gram до 5, min_df, max_df);
- число комбинаций в `GridSearchCV`;
- характеристики машины (CPU, объём оперативной памяти).

В реальном эксперименте:

- **Logistic Regression** + `GridSearchCV` — обучение занимает порядка нескольких минут;
- **Linear SVM (calibrated)** — сопоставимое время с логистической регрессией (несколько минут);
- **Multinomial Naive Bayes** — обучается за секунды.

Важно: это обучение делается **один раз**; в Streamlit-приложении модели только загружаются из файлов `.joblib`, поэтому онлайн-предсказание происходит практически мгновенно.

Фрагмент кода обучения (Logistic Regression)

```
# TF-IDF векторизация
vec = TfidfVectorizer( ngram_range=(1, 5), min_df=5, max_df=0.9 )
X_train_vec = vec.fit_transform(X_train)
X_test_vec  = vec.transform(X_test)
logreg = LogisticRegression(max_iter=2000)

param_grid_lr = {
    "C": [0.25, 0.5, 1.0, 2.0],
    "class_weight": [None, "balanced"],
}
gs_lr = GridSearchCV(
    logreg, param_grid_lr, scoring="f1", cv=3, n_jobs=-1
)
gs_lr.fit(X_train_vec, y_train)
best_lr = gs_lr.best_estimator_
```

Для SVM и Naive Bayes используется аналогичная схема с `GridSearchCV` и последующей оценкой на тестовой выборке.

Метрики качества

Используем стандартные метрики бинарной классификации:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1} = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Почему F1 важнее одной Accuracy?

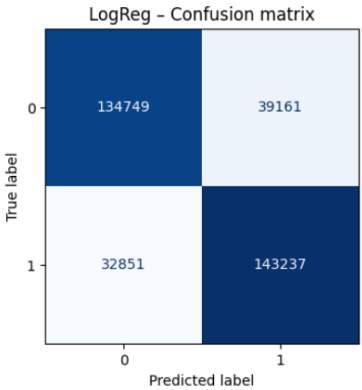
- в корпусе классы распределены не идеально равномерно;
- F1 учитывает баланс между полнотой и точностью для «позитива».

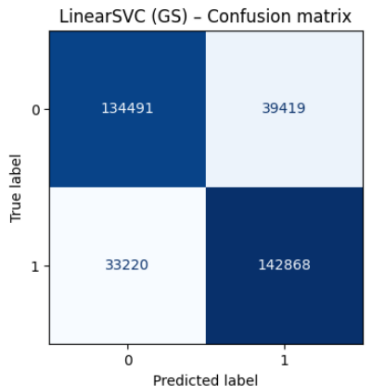
Результаты (таблица)

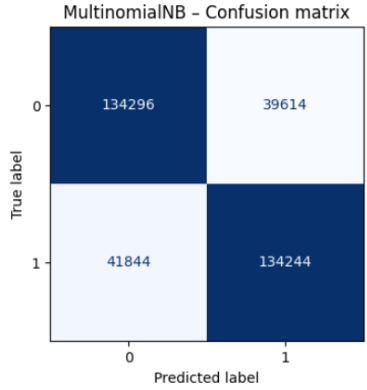
Модель	Accuracy	Precision_w	Recall_w	F1_w
Logistic Regression	0.794	0.794	0.794	0.794
Linear SVM (calibrated)	0.790	0.791	0.790	0.790
Multinomial Naive Bayes	0.767	0.767	0.767	0.767

Лучшей стала **Logistic Regression**, Linear SVM даёт очень близкий результат.

Confusion matrix для моделей

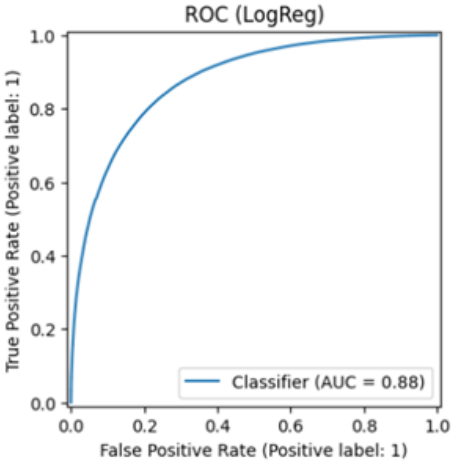


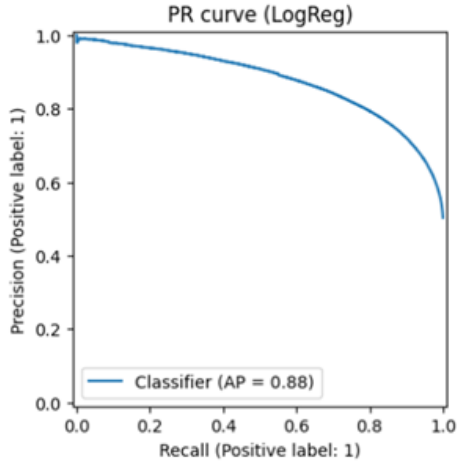




Слева направо: Logistic Regression, Linear SVM, Multinomial Naive Bayes.

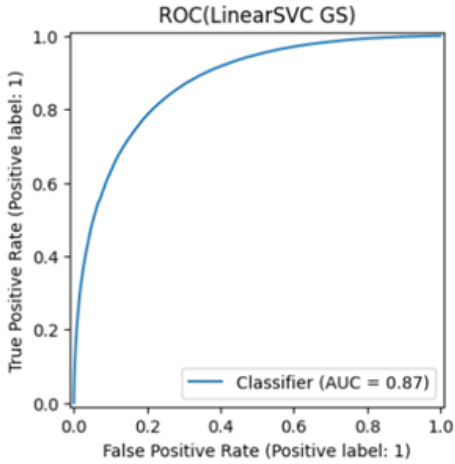
ROC и PR-кривые

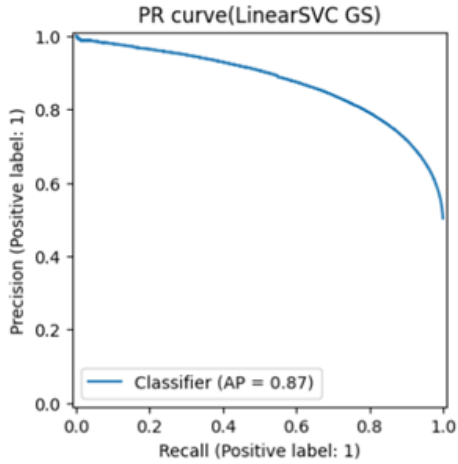




Каждый блок: слева ROC-кривая (AUC), справа Precision–Recall (AP). LogReg и SVM показывают AUC ≈ 0.87–0.88, Naive Bayes чуть хуже.

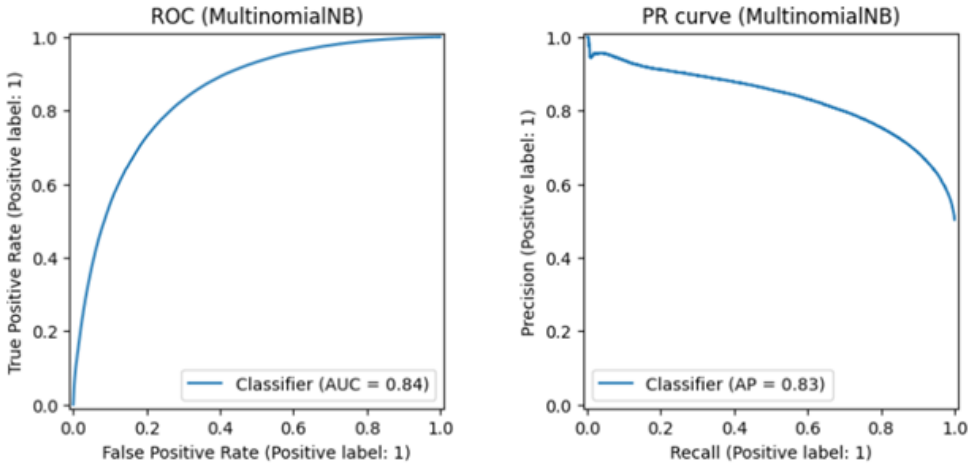
ROC и PR-кривые





Каждый блок: слева ROC-кривая (AUC), справа Precision–Recall (AP). LogReg и SVM показывают AUC ≈ 0.87–0.88, Naive Bayes чуть хуже.

ROC и PR-кривые



Каждый блок: слева ROC-кривая (AUC), справа Precision–Recall (AP). LogReg и SVM показывают AUC ≈ 0.87–0.88, Naive Bayes чуть хуже.

Выбор итоговой модели

По результатам экспериментов в ноутбуке:

- Logistic Regression и Linear SVM показывают близкие значения F1 ($\approx 0{,}79$);
- Multinomial Naive Bayes заметно уступает им по качеству.

Лучше всего "в среднем" себя ведёт **Logistic Regression**, потому что:

- даёт наивысший F1 на тестовой выборке;
- даёт вероятности «из коробки» (удобно для интерфейса);
- веса признаков интерпретируемы (можно анализировать важные слова).

Экспорт моделей

Лучшие модели и TF-IDF-векторизатор сохраняются в виде артефактов:

```
joblib.dump(vec,       "artifacts/tfidf_vectorizer.joblib")
joblib.dump(best_lr,   "artifacts/logreg_best.joblib")
joblib.dump(best_lsvc, "artifacts/linear_svc_best.joblib")
joblib.dump(best_mnb,  "artifacts/mnb_best.joblib")
```

Эти файлы затем загружаются в веб-приложении для онлайн-предсказаний.

Streamlit-приложение

Веб-интерфейс на `Streamlit`:

- загрузка TF-IDF и трёх моделей из папки `artifacts`;
- выбор модели (*LogReg / Linear SVM / Naive Bayes*);
- ввод текста пользователем;
- предобработка: токенизация, стоп-слова, стемминг;
- вывод:
  - вероятность позитивного / негативного класса;
  - метка настроения;
  - текст до и после предобработки;
  - история последних запросов и вкладка «Сравнение моделей».

Такое приложение демонстрирует работу модели «вживую».

Выводы

Что сделано:

- реализован полный pipeline: от отзывов до веб-приложения;
- проведено обучение и сравнение трёх классических моделей;
- Logistic Regression показала наилучший F1, Linear SVM — сопоставимое качество;
- модель успешно вынесена в Streamlit-интерфейс.

Использованные материалы (1/2)

Датасеты

- Kaggle Twitter Sentiment Dataset (`train.csv`);
- Sentiment140: training_noemoticon (1.6M размеченных отзывов).

Библиотеки

- `scikit-learn`: TfidfVectorizer, LogisticRegression, LinearSVC, CalibratedClassifierCV, MultinomialNB, GridSearchCV, метрики;
- `NLTK`: английские стоп-слова, PorterStemmer;
- `Streamlit`: веб-интерфейс для демонстрации модели;
- `matplotlib`, `seaborn`: построение ROC/PR-кривых и матриц ошибок.

Использованные материалы (2/2)

- Предварительная обработка данных в машинном обучении
- Sentiment Analysis
- Статьи «Классификация текстов и анализ тональности» и «Оценка качества в задачах классификации», портал NEERC: `https://neerc.ifmo.ru/wiki/`
- Статья «Автоматический анализ тональности текстов: проблемы и методы», журнал «Информационные процессы», MathNet: `https://www.mathnet.ru/`
- Статья «Анализ тональности текста методами машинного обучения», портал CyberLeninka: `https://cyberleninka.ru/`