ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ» (ФИНАНСОВЫЙ УНИВЕРСИТЕТ)

Департамент анализа данных, принятия решений и финансовых технологий

Дисциплина: «Технологии анализа данных и машинное обучение»

Направление подготовки: «Прикладная математика и информатика»

Профиль: «Анализ данных и принятие решений в экономике и финансах»

Факультет информационных технологий и анализа больших данных

Форма обучения очная
Учебный 2020/2021 год, 5 семестр

Курсовая работа на тему:

«Машинное обучение в задачах предсказания оттока клиентов»

Допущен к защите 16.12.2020

Выполнила(а):

студент группы ПМ18-3

Тумасян Д. А.

Руководитель:

к. ф.-м. н., доцент

Мейханаджян Л. А.

Москва 2020

Содержание

Введение

С постепенной диджитализацией различных сфер общественной жизни, экономики, промышленности появилась острая потребность в хранении данных, их обработке и анализе. И если под словом анализ раньше зачастую понималось извлечение каких-либо выводов о результатах деятельности той или иной сферы за предшествующие периоды с целью корректировки последующих действий в управлении и принятии решений, которые не всегда могут оказаться эффективными, то сейчас анализ больше имеет дело с машинным обучением, которое позволяет не только избегать задействование человеческого фактора, но и предсказывать и довольно точно прогнозировать результаты каких-либо предполагаемых принятых решений.

Одной из таких задач, которую можно рассмотреть в рамках методов машинного обучения – задача предсказания оттока клиентов (*customer churn prediction*). Смысл данной задачи заключается в том, чтобы предсказать по набору параметров, описывающих клиентов компании, откажутся ли они от услуг последней, т. е. склонны они к оттоку или нет. Таким образом, задача состоит в том, чтобы определить принадлежность клиента к одному из классов бинарного множества {0, 1}, где 0 – клиент склонен к оттоку, 1 – клиент не склонен к оттоку. Иногда, в подобной задаче, при более тщательном ее изучении, можно обнаружить два типа оттока - договорный (*contractual*) и недоговорный (*non-contractual*). Первый сопровождается довольно очевидными паттернами поведения – клиент четко определяет свой отток действиями, например, отпиской от сервиса. В то время как недоговорный отток не обозначается четким образом со стороны клиента. Такой отток можно обнаружить через резкое снижение активности пользователя, при этом условный договор пользования им не расторгается. Ярким примером является пользование онлайн-сервисами какой-либо компании, когда признаком оттока будет являться время последнего взаимодействия клиента с сервисом. К сожалению, второй тип оттока более распространен, что усложняет задачу предсказания.

Цель данной работы – изучить на реальных данных поведение клиентов и попытаться построить модель машинного обучения с максимально высокой точностью, позволяющую выявлять отток клиентов, что в свою очередь, может позволить компании предпринять меры по удержанию клиентов, что является ключевой бизнес-целью для любой компании.

В ходе работы будет рассмотрен датасет, предоставленный в открытом доступе на ресурсе kaggle.com, состоящий из реальных данных о клиентах компании – банка, на которых будут обучены 7 классификаторов, использующие разные алгоритмы машинного обучения (Decision Tree, K-nearest Neighbors, Random Forest, Logistic Regression, SVC (Support Vector Classification), Multi-Layer Perceptron, Gradient Boosting).

Модели будут описаны и обучены на языке программирования Python (версия 3.7.1), с привлечением библиотек numpy, pandas, scikit-learn, а также пакета matplotlib для визуализации данных.

1. Предварительная обработка данных

Для исследования используется датасет, загруженный с ресурса kaggle.com, содержит данные о клиентах, в том числе об их статусе оттока. Таблица содержит данные 10000 клиентов, которые описываются следующими атрибутами:

- **CustomerID** - уникальный номер клиента
- **Surname** - фамилия
- **CreditScore** – балл клиента по результату кредитного скоринга
- **Geography** – страна проживания
- **Gender** - пол
- **Age** - возраст
- **Tenure** – срок пребывания клиента с компанией
- **Balance** – баланс на счету клиента
- **NumOfProducts** – количество продуктов компании, которые использует клиент
- **HasCrCard** – наличие кредитной карты
- **IsActiveMember** – статус активности клиента
- **EstimatedSalary** – приблизительная зарплата
- **Exited** – статус оттока

Также, заметим, что в таблице помимо количественных и порядковых признаков, содержатся категориальные – ‘Geography, ‘Gender’. Их, конечно, можно закодировать числовыми значениями, однако в данном случае воспользуемся функцией `get_dummies` из пакета pandas, которая для каждого значения атрибута создаст столбец, значения которого будут принадлежать множеству $\{0,1\}$ – обозначая принадлежность клиента к тому или иному признаку.

С помощью команды `df.isna.sum()`подсчитаем количество пропущенных данных в каждом столбце. Заметим, что такие столбцы с NaN значениями

отсутствуют, что указывает на довольно чистый и равномерный характер данных в датасете.

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
    dtype: int64
```

Построим также тепловую карту, основанную на корреляционной матрице признаков используемого датасета:

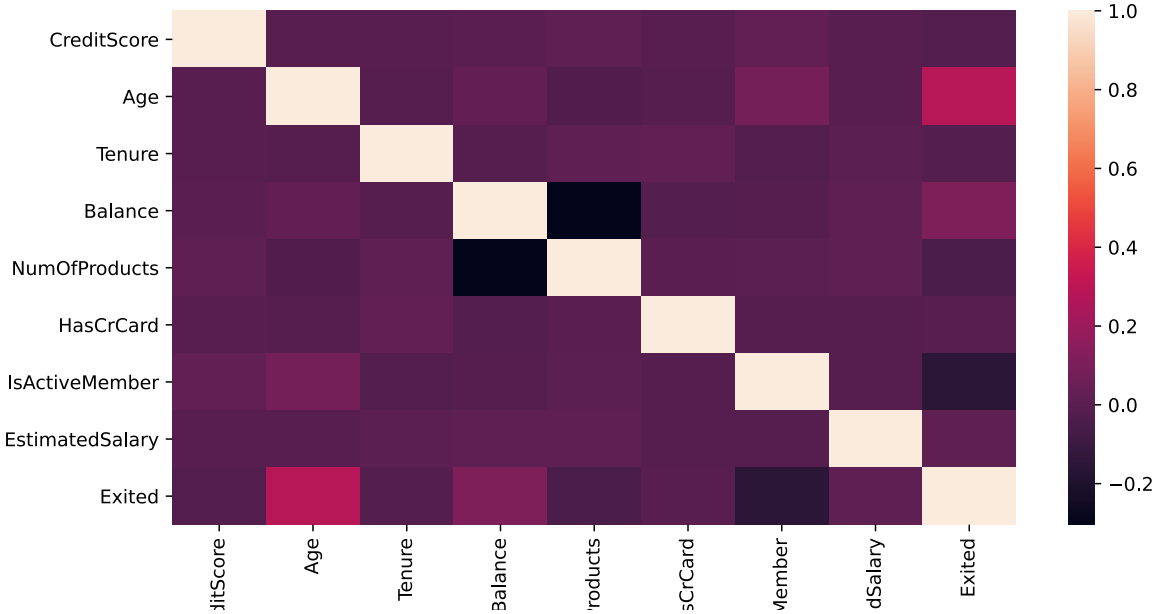


Рис. 1. Тепловая карта корреляционной матрицы атрибутов

Ни одна из пар признаков не находится в ощутимой корреляции. Таким образом, исходный датасет не подлежит очистке и готов к обработке и обучению.

Можно заметить, что в таблице содержаться поля с уникальными значениями (‘CustomerID’, ‘Surname’), которые не несут никакой смысловой нагрузки в случае обучения, их можно удалить из таблицы.

2. Обучение на разных моделях

Как и было указано раннее, данные будут обучены на 7 разных классификаторах. Для оценки их эффективности и качества стоит ввести метрики и дать им характеристику.

2.1. Метрики моделей

Любые метрики, описывающие качество обучения моделей, опираются на базовые понятия об ошибках при принятии гипотез, если говорить языком математической статистики.

Выделяют ошибки первого и второго рода.

Ошибка I рода (или false positive, FP) – ситуация, когда принято неправильное решение о принадлежности объекта классу ‘1’ (TRUE).

Ошибка II рода (или false negative, FN) – ситуация, когда принято неправильное решение о принадлежности объекта классу ‘0’ (FALSE).

На основе этого составим так называемую матрицу ошибок. Положим Y – истинная метка класса, Y^* – предсказанная алгоритмом. Тогда, возникают четыре ситуации.

	$Y^* = 0$	$Y^* = 1$
$Y = 0$	TN (true negative)	FP (false positive)
$Y = 1$	FN (false negative)	TP (true positive)

Дальше символами TN, FP, FN, TP будем обозначать долю соответствующих результатов, выданных алгоритмом. На основе этих долей и высчитываются основные метрики моделей.

Всего рассмотрим 5 метрик – accuracy, recall, precision, F-меру и AUC ROC.

Accuracy вычисляется как доля всех правильных ответов, выданных алгоритмом, т. е.:

$$\mathbf{accuracy} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

Основной недостаток данной метрики – она не учитывает возможные изначальные неравные размеры классов.

Precision указывает на долю объектов с положительной, по мнению алгоритма, меткой из всех объектов с такой меткой:

$$\mathbf{precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Также, существует довольно похожая метрика – **recall**, которая вычисляется как доля объектов выборки с положительной, по мнению алгоритма, меткой из всех объектов, которые на самом деле относятся к положительному классу:

$$\mathbf{recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Для нашей задачи лучше всего подходят последние две метрики из-за безразличности к неравенству классов, ведь при оценке качества модели мы полагаем, что предпочтительнее определить клиентов, которые не подвержены оттоку как утекающих, нежели наоборот.

Однако, на практике выясняется, что невозможно найти абсолютно всех клиентов, которые будут подвержены оттоку и только их. Для этого можно задать некий порог, при котором уходящий клиент для нас значим, и таким образом попытаться найти некий баланс между precision и recall.

Одна из таких метрик, позволяющая аккумулировать «полезность» как полноты, так и точности является **F-мера.** Задается она следующим образом (в общем виде):

$$\boldsymbol{F_\beta} = (1 + \beta^2)\frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

β – весовой коэффициент точности. При $\beta = 1$ имеем среднее гармоническое от полноты и точности. Вообще говоря, $\lim\limits_{\substack{\text{precision}\to 1 \\ \text{recall}\to 1}} F_\beta = 1$.

Еще одна метрика, **AUC-ROC** (*Area Under Curve of Receiver Operating Characteristic curve*, или площадь под графиком кривой ошибок) пригождается при отображении объектов вещественных классов в бинарное множество {0, 1} (например, если ранжирование классификатором происходит через вычисление вероятности принадлежности объекта к тому или иному классу). В таком случае возникает естественный вопрос – какое значение является пороговым, переходным - из 0 в 1, ведь вполне очевидное значение 0,5 не подходит по тем же причинам, что и метрика accuracy, когда приходится иметь дело с неравными классами? Для такой оценки строится кривая ошибок, представляющая собой кривую из точки (0,0) в точку (1,1), в системе координат (FPR, TPR), где:

$$\boldsymbol{FPR}\ (\boldsymbol{false\ positive\ rate}) = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\boldsymbol{TPR}\ (\boldsymbol{true\ positive\ rate}) = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Каждая точка на графике представляет собой значение определенного порога. Качество работы алгоритма повышается с максимизацией TPR и минимизацией FPR, т. е. график кривой должен стремиться в точку (0,1). При этом будет расти площадь под графиком, что и представляет собой AUC ROC. Метрика AUC ROC довольно устойчива к несбалансированности между классами.

2.2. Decision Tree

Decision tree (или, решающее дерево) представляет собой алгоритм, принцип работы которого имитирует структуру дерева, где в каждой вершине (т. н. «лист») проверяется удовлетворение какого-либо атрибута конкретному условию, на основе чего алгоритм продолжает строить подобные классифицирующие вершины, до тех пор пока все из них не окажутся терминальными с конкретной меткой класса.

Ниже продемонстрированы результаты работы DecisionTreeClassifier:

	precision	recall	f1-score	support
0	0.88	0.88	0.88	2416
1	0.51	0.52	0.51	584
accuracy			0.81	3000
macro avg	0.69	0.70	0.70	3000
weighted avg	0.81	0.81	0.81	3000
ROC AUC score	**0.83**			

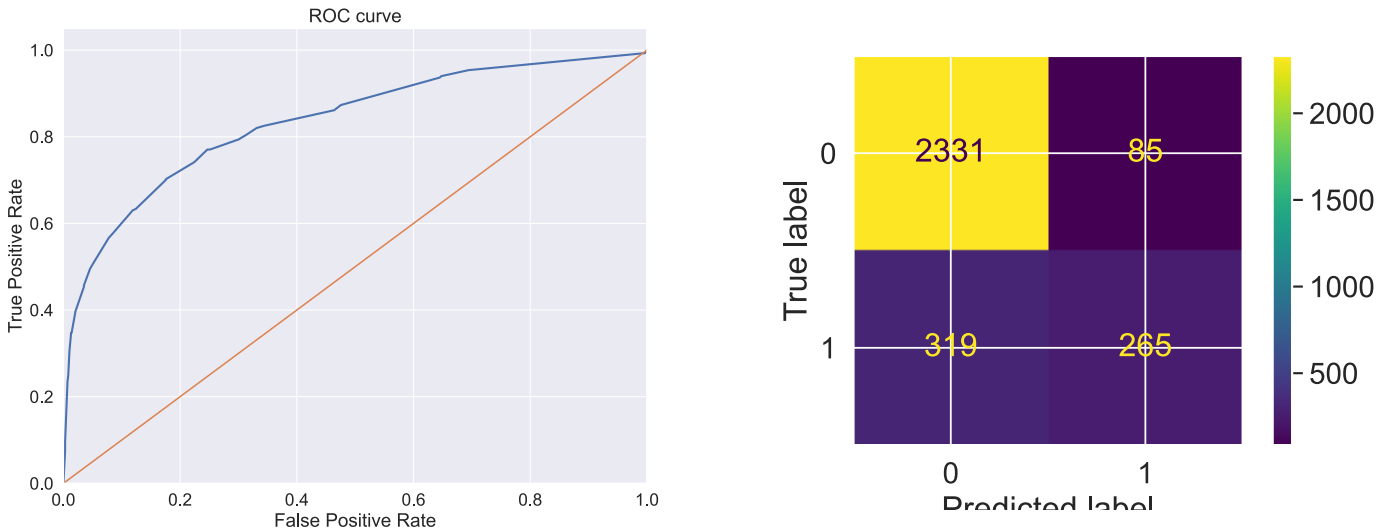


Рис. 2. Кривая ошибок и матрица ошибок для DecisionTreeClassifier

2.3. K-nearest Neighbors

Принцип работы **K-nearest Neighbors** (или метод k-ближайших соседей) также довольно примитивен. Данный классификатор определяет метку объекта тестовой выборки, основываясь на том, объекты какого класса превалируют во множестве k-ближайших соседей данного тестового образца. При этом метрику (по умолчанию – расстояние Миньковского) и ее тип на выборке для определения расстояния между объектами можно задать заранее. Продемонстрируем результат работы KNeighborsClassifier при заданных по умолчанию гиперпараметрах:

	precision	recall	f1-score	support
0	0.81	0.93	0.87	2416
1	0.25	0.09	0.14	584
accuracy			0.77	3000
macro avg	0.53	0.51	0.50	3000
weighted avg	0.70	0.77	0.73	3000
ROC AUC score	**0.54**			

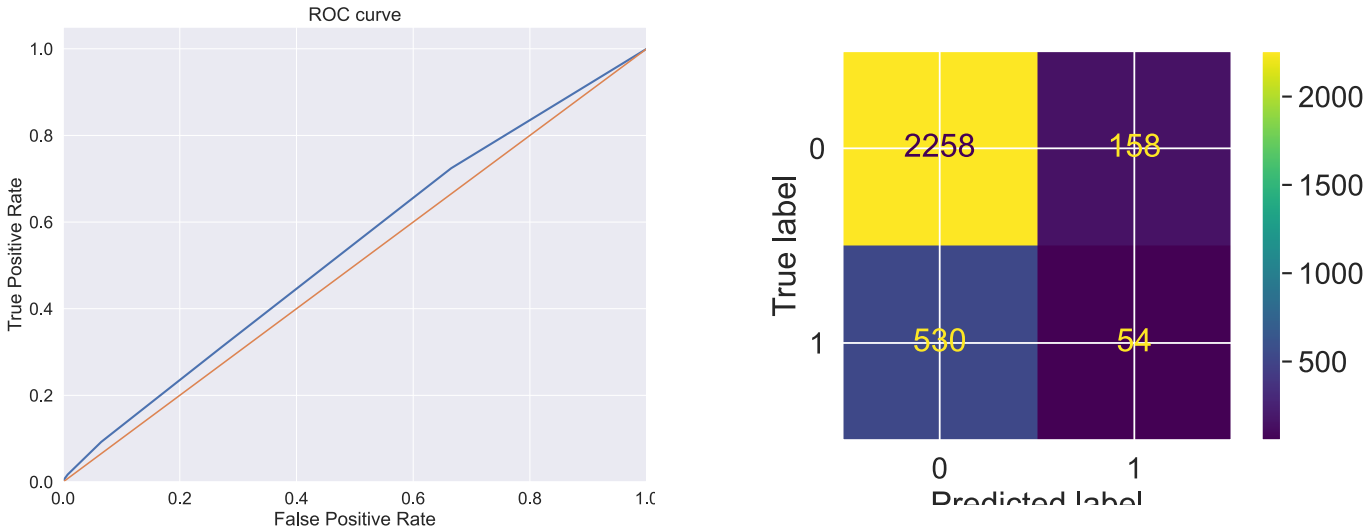


Рис. 3. Кривая ошибок и матрица ошибок для KNeighborsClassifier

2.4. Random Forest

Случайный лес или так называемый **Random Forest** – ансамбль методов машинного обучения, которые основывается на построении многочисленных решающих деревьев во время обучения. Такая особенность алгоритма призвана решать проблему переобучения решающих деревьев без прунинга (отсечения, например с помощью ввода параметра `max_depth`). Классификатор выдает ответ – метку, которая чаще всех встречается среди всех меток, выданных деревьями случайного леса.

Результат работы алгоритм RandomForestClassifier:

	precision	recall	f1-score	support
0	0.88	0.96	0.92	2416
1	0.76	0.46	0.57	584
accuracy			0.87	3000
macro avg	0.82	0.71	0.74	3000
weighted avg	0.86	0.87	0.85	3000
ROC AUC score	**0.85**			

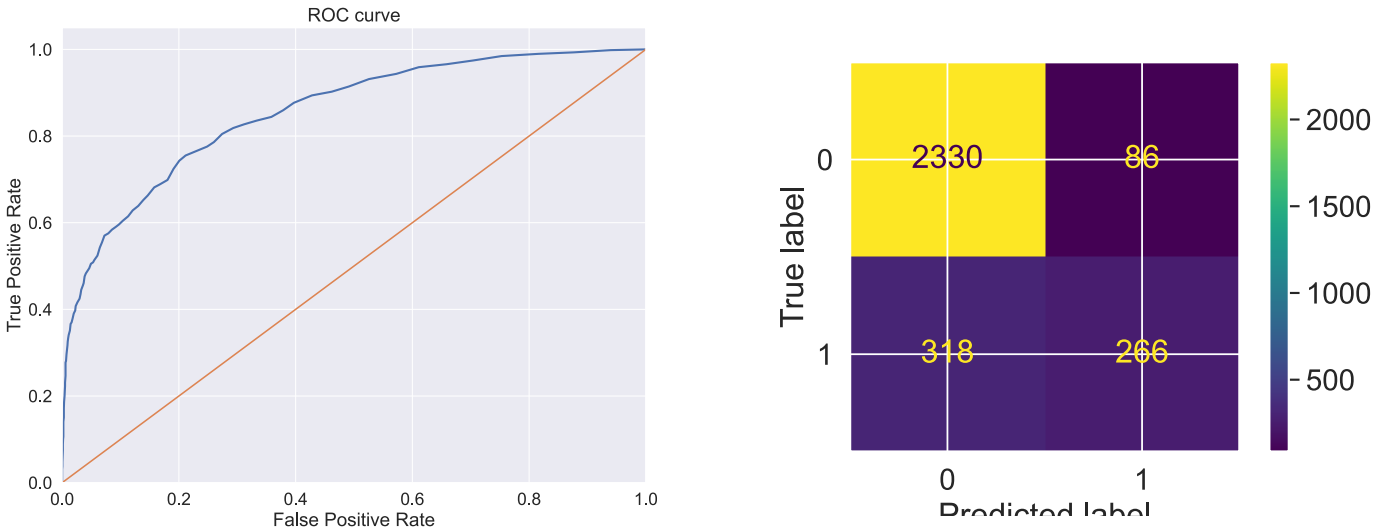


Рис. 4. Кривая ошибок и матрица ошибок для RandomForestClassifier

2.5. Logistic Regression

Логистическая регрессия, несмотря на свое название, является алгоритмом классификации, который оценивает апостериорную вероятность принадлежности объектов выборки одному из классов и на основе этого ранжирует их.

Допустим, объекты исследуемой выборки X описываются n числовыми признаками, $\mathcal{F}_j: X \to \mathbb{R}, j = \overline{1,n}$, а $Y -$ множество меток. В нашем случае $Y = \{0, 1\}$.

Предположим, $X_{train} = \{(x_1, y_1), \dots, (x_k, y_k)\}$ – обучающая выборка размерности k, состоящая из пар «объект-метка». Логистическая регрессия строит классификатор $a: X \to Y$, который имеет вид:

$$a(x, w) = \text{sign}\left(\sum_{j=1}^{n} w_j \mathcal{F}_j(x) - w_0\right) = \text{sign}\langle x, w\rangle$$

где w_j — вес j-го признака, w_0— порог принятия решения, $w = \{w_0, w_1, \dots, w_0\}$ — вектор весов. Логистическая регрессия подбирает такой вектор весов на основе обучающей выборки, которая бы минимизировала функцию потерь. После нахождения такого вектора алгоритм уже способен высчитать апостериорные вероятности принадлежности объектов одному из двух классов:

$$\mathbb{P}(y|x) = \sigma(y\langle x, w\rangle), \qquad y \in Y$$

где $\sigma(z) = \frac{1}{1+e^{-z}}$ – сигмоид или так называемая *логистическая функция*.

Результат работы LogisticRegression:

	precision	recall	f1-score	support
0	0.81	0.97	0.89	2416
1	0.44	0.08	0.14	584
accuracy			0.80	3000
macro avg	0.63	0.53	0.51	3000
weighted avg	0.74	0.80	0.74	3000
ROC AUC score	**0.67**			

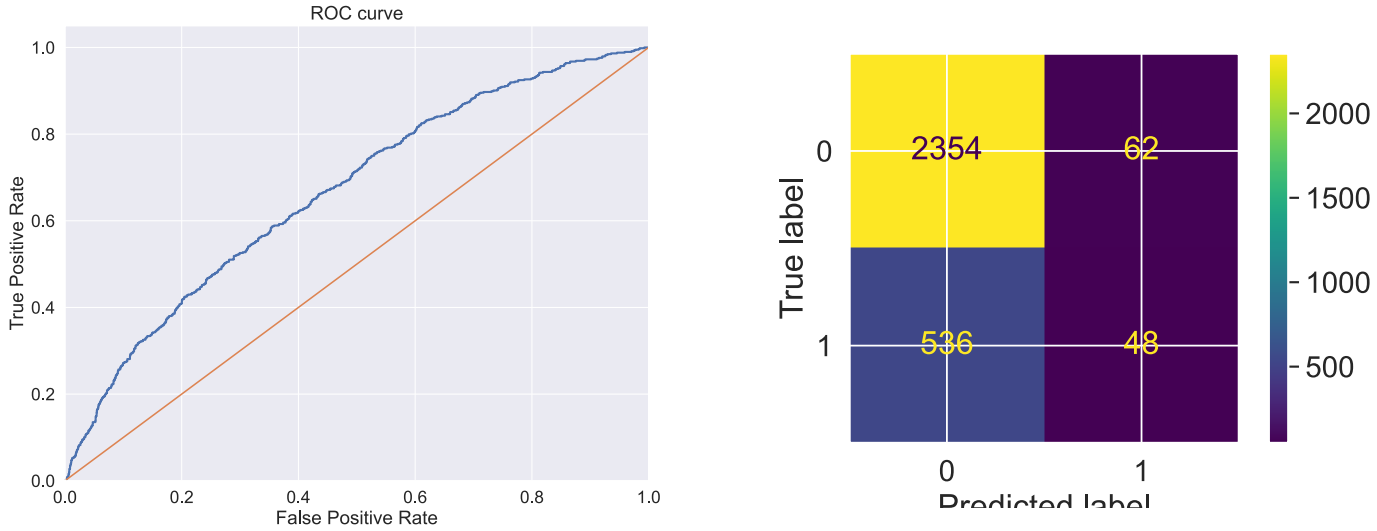


Рис. 5. Кривая ошибок и матрица ошибок для LogisticRegression

2.6. SVM (Support Vector Machine)

Метод опорных векторов (**SVM, Support Vector Machine**) алгоритм машинного обучения с учителем. Все объекты исследуемой выборки описываются n числовыми признаками, тем самым каждому объекту можно сопоставить точку в $\mathbb{R}^n$. Задача заключается в нахождении уравнения разделяющей гиперплоскости размерности $n-1$, благодаря которой стало бы понятно, что объекты, находящуюся по одну сторону от нее принадлежат одному классу и наоборот.

Таких гиперплоскостей, которые бы верно разбивали объекты при обучении на два класса, может быть довольно много. *Оптимальной разделяющей гиперплоскостью* будет считаться та, расстояния от которой до двух ближайших точек обоих классов будут максимальны.

Результат работы SVC (Support Vector Classification):

	precision	recall	f1-score	support
0	0.81	1.00	0.89	2416
1	0.00	0.00	0.00	584
accuracy			0.81	3000
macro avg	0.40	0.50	0.45	3000
weighted avg	0.65	0.81	0.72	3000
ROC AUC score	**0.54**			

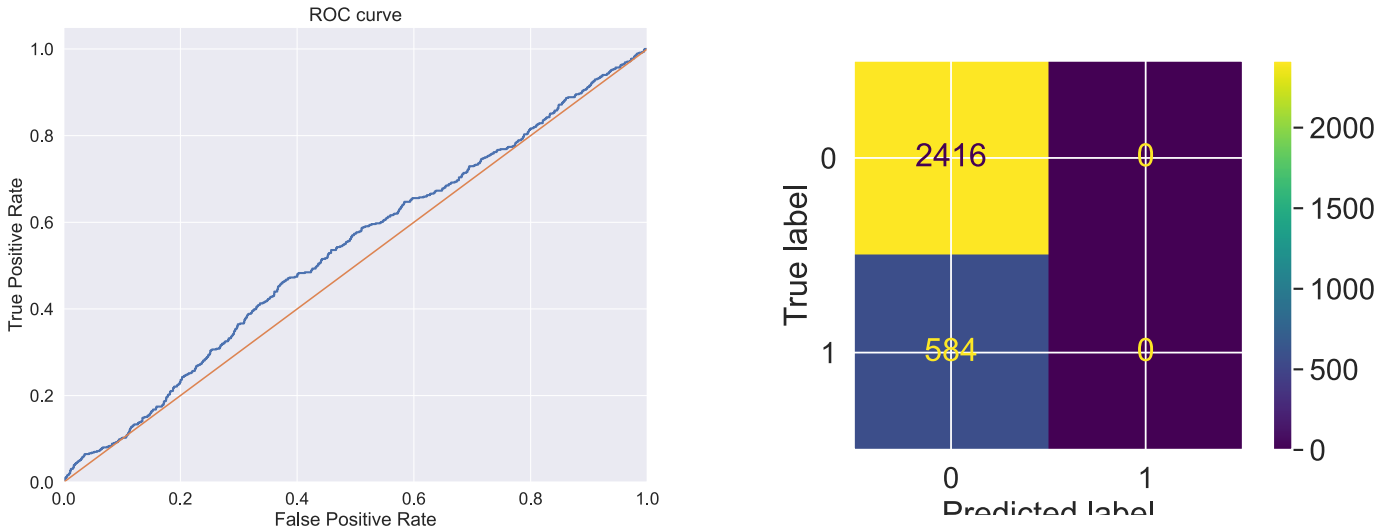


Рис. 6. Кривая ошибок и матрица ошибок для SVM

2.7. MLP (Multi-layer Perceptron)

Многослойный перцептрон (**Multi-layer Perceptron, MLP**) – модель искусственной нейронной сети прямого распространения, которая преобразует набор входящих данных в набор приемлемых результатов вычислений. MLP обучается интерактивно, так как на каждом временном шаге вычисляются частные производные функции по параметрам модели для обновления параметров.

Результат работы MLPClassifier:

	precision	recall	f1-score	support
0	0.93	0.11	0.19	2416
1	0.21	0.97	0.34	584
accuracy			0.27	3000
macro avg	0.57	0.54	0.27	3000
weighted avg	0.79	0.27	0.22	3000
ROC AUC score	**0.61**			

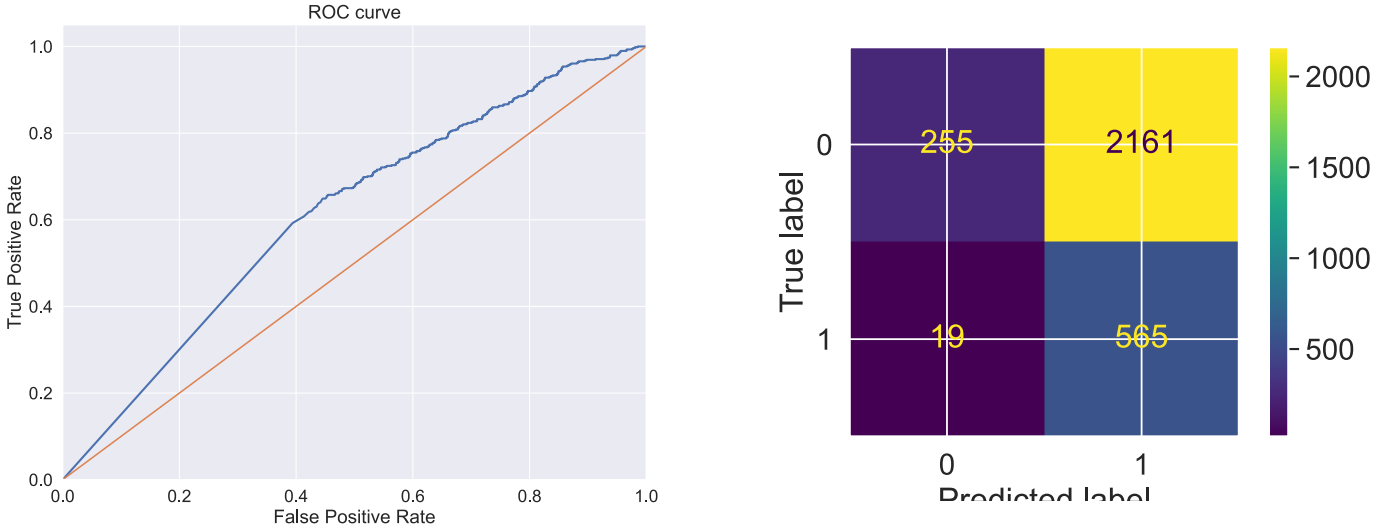


Рис. 7. Кривая ошибок и матрица ошибок для MLPClassifier

2.8. Gradient Boosting

Градиентный бустинг (**Gradient Boosting**) помогает восстановить зависимость вида $y = f(x)$ для пар из набора $X_{train} = \{(x_1, y_1), \dots, (x_k, y_k)\}$ в виде приближения $\hat{f}(x)$ при условии минимизации функции потерь $L\big(y, f(x)\big)]$, где $y \approx \hat{f}(x) = f\big(x, \hat{\theta}\big)$. С помощью следующего алгоритма подбирается оптимальный набор параметров $\hat{\theta}$:

1. Задается начальное приближение набора параметров $\hat{\theta} = \hat{\theta}_0$
2. Для каждой итерации $n = 1, \dots, M$ повторяются следующие шаги:
   - вычисляется градиент функции потерь $\nabla L_\theta\big(\hat{\theta}\big)$ при текущей на данном этапе аппроксимации $\hat{\theta}$

$$\nabla L_\theta\big(\hat{\theta}\big) = \left[\frac{\partial L(y, f(x, \theta)}{\partial \theta}\right]_{\theta = \hat{\theta}}$$

   - обновляется значение очередного итеративного приближения

$$\hat{\theta}_n = -\nabla L_\theta\big(\hat{\theta}\big)$$

   - обновляется приближение параметров $\hat{\theta} \leftarrow \hat{\theta}_n + \hat{\theta} = \sum_{i=0}^{n} \hat{\theta}_i$

$$\hat{\theta} \leftarrow \hat{\theta}_n + \hat{\theta} = \sum_{i=0}^{n} \hat{\theta}_i$$

3. Сохраняется итоговое приближения посредством суммирования приближений за счет спуска на каждой итерации

$$\hat{\theta} = \sum_{i=0}^{M} \hat{\theta}_i$$

Продемонстрируем работу GradientBoostingClassifier:

	precision	recall	f1-score	support
0	0.89	0.97	0.92	2416
1	0.77	0.48	0.60	584
accuracy			0.87	3000
macro avg	0.83	0.73	0.76	3000
weighted avg	0.86	0.87	0.86	3000
ROC AUC score	**0.87**			

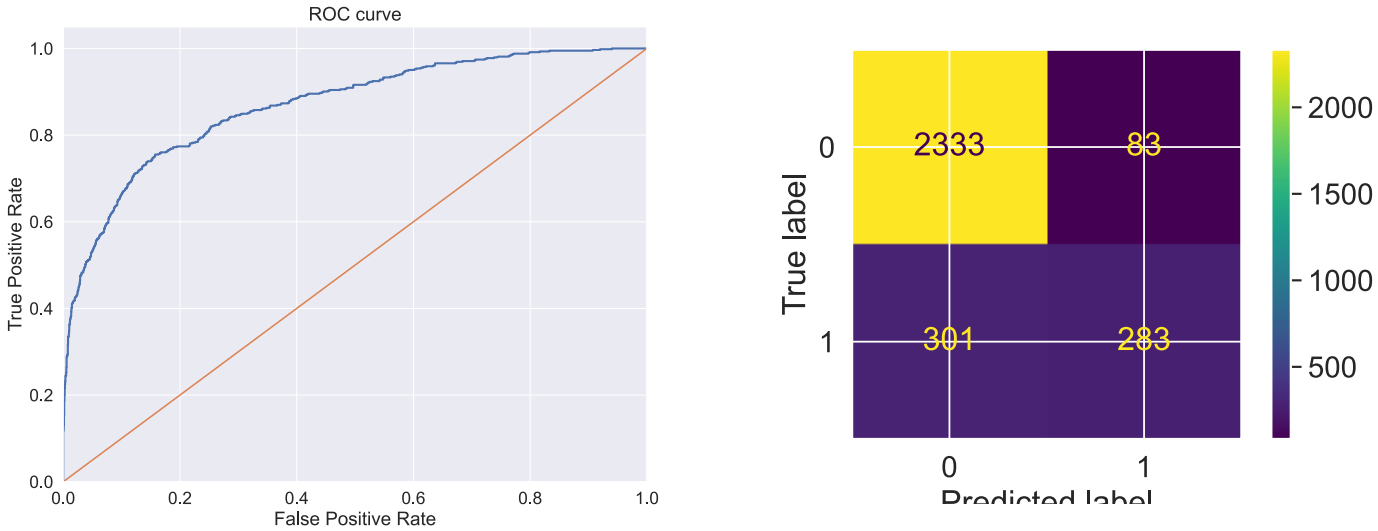


Рис. 8. Кривая ошибок и матрица ошибок для GradientBoostingClassifier

2.9. Анализ и сравнение работы моделей

После обучения семи разных классификаторов необходимо выявить самый лучший по качеству обучения. Ниже представлена краткая сводка-таблица со значениями метрик для каждого из алгоритмов для метки ‘0’, так как предпочтительнее исследовать работу алгоритма на утекающих клиентах. Стоит заметить, что при данном анализе игнорируется метрика accuracy ввиду своей бесполезности в контексте данной задачи:

	recall	precision	F-мера	ROC AUC
DecisionTreeClassifier	0,87	0,88	0,88	0,83
KNeighborsClassifier	0,81	0,93	0,87	0,54
RandomForestClassifier	*0,88*	*0,96*	***0,92***	*0,85*
LogisticRegression	0,81	0,97	0,89	0,67
SVC	0,81	1,00	0,89	0,54
MLPClassifier	0,93	0,11	0,19	0,61
GradientBoostingClassifier	**0,89**	**0,97**	**0,92**	**0,87**

Таблица 1. Значения метрик для классификаторов

Основной упор необходимо делать на метрики recall и accuracy, поэтому проще будет отобрать наиболее качественную модель в первую очередь по F-мере. Таких моделей оказалось всего две – RandomForestClassifier и GradientBoostingClassifier. Однако, по остальным значениям метрик выигрывает градиентный бустинг.

3. Усовершенствование наиболее перспективной модели

В данном разделе описаны различные методы усовершенствования работы выбранной ранее наиболее перспективной модели и результаты их применения.

Методов улучшения качества работы алгоритмов множество: понижение размерности для создания суррогатных признаков, изменения методов и порядка предобработки данных, нормализация и регуляризация данных, ансамблирование нескольких моделей машинного обучения, и, наконец, самый распространенный – подбор гиперпараметров.

Для начала стандартизируем данные в обучающих и тестовых выборках с помощью модуля `sklearn.preprocessing` в котором содержится метод **StandardScaler**. Он преобразует данные по следующей формуле:

$$\tilde{X} = \frac{X - m}{\sigma}$$

где m – математическое ожидание по выборке, а σ – ее стандартное отклонение. В итоге все числовые данные соответствуют условию $\tilde{X} \in [0,1]$.

Изменим также объем самой тестовой выборки. Подберем оптимальную величину параметра `test_size` в `train_test_split`, при котором бы получали максимальный ROC AUC. При `test_size` = 0.17 получаем ROC AUC = 0,872068 вместо 0.868282.

Принимая во внимания изменения, указанные выше, запустим поиск оптимальных гиперпараметров (а именно, `loss, learning_rate, subsample, n_estimators, max_features`) по сетке **GridSearchCV** со следующими диапазонами:

```
params = {'loss': ['deviance', 'exponential'],
          'learning_rate' : [i/1000 for i in range(90,100)],
          'subsample': [0.9],
          'n_estimators': range(85, 91),
          'max_features': ['auto', 'log2', 'sqrt', None]}
```

GridSearchCV выдал следующие оптимальные гиперпараметры и значение ROC AUC для GradientBoostingClassifier с полученными параметрами:

```
learning_rate: 0.096,
loss: 'deviance',
max_features: 'auto',
n_estimators: 87,
subsample: 0.9
ROC AUC: 0.873194
```

Очевидно, качество обучения модели слегка улучшилось (0,873194 в сравнении с изначальным 0,868282).

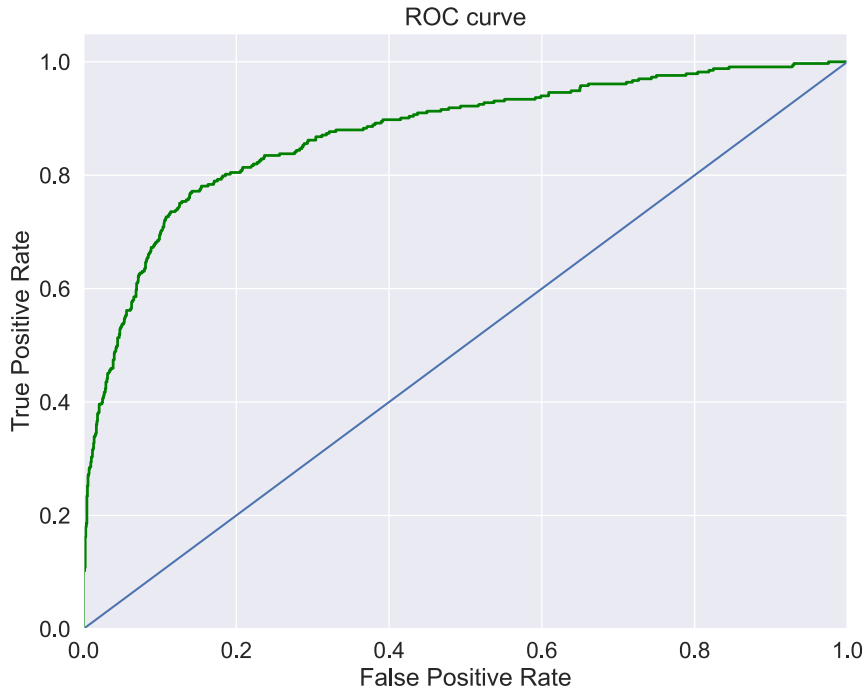


Рис. 9. Кривая ошибок для модифицированной GradientBoostingClassifier

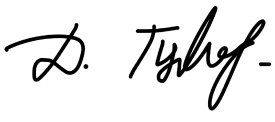
При предварительном анализе данных было выявлено, что корреляция между парами атрибутов несущественны, их удаление скорее приведет к ухудшению качества обучения. По этой же причине пропадает необходимость в рассмотрении `feature_importances_`.

Заключение

В ходе проделанной курсовой работы были достигнуты упомянутые во введении цели. После тщательного анализа и очистки исходного набора данных, их обработки и подготовки к обучению, они были протестированы на 7 алгоритмах-классификаторах. Были рассмотрены принципы работы каждого из них, что показало в свою очередь, что довольно сложные алгоритмы не всегда выдают хорошее качество предсказания и обучения.

Улучшенный классификатор GradientBoostingClassifier обучился довольно точно предсказывать клиентов, подверженных оттоку. Однако, такая точность предсказания все равно недостаточна для прикладных задач в таких областях, как e-commerce, банки, услуги сотовой связи.

Тем не менее, данная задача довольно актуально в сфере услуг, особенно сейчас, когда тяжелые экономические последствия пандемии также влияют на отток клиентов и, соответственно, новые факторы могут влиять на решение клиентов остаться или же расторгнуть договор.



Список литературы

1. Библиотека scikit-learn: [Электронный ресурс]// URL: https://scikit-learn.org/stable/ (Дата обращения: 28.11.2020)
2. *Коэльо Л.П., Ричарт В.* Построение систем машинного обучения на языке Python. 2016. 305 с.
3. *Hastie T., Tibshirani R., Friedman J.* The Elements of Statistical Learning. Springer, 2014.
4. Андреас Мюллер, Введение в машинное обучение с помощью Python [Руководство для специалистов по работе с данными] / Андреас Мюллер, Сара Гвидо, – Москва, 2016-2017. – 393 с.
5. https://www.coursera.org/learn/vvedenie-mashinnoe-obuchenie/home/welcome
6. http://www.machinelearning.ru/wiki/index.php?title=Проверка_статистических_гипотез

Приложение

Приложение 1

Процессор и его тактовая частота: Intel Core i5-8257U CPU @ 1,4 GHz (Turbo Boost up to 3,9 GHz)

Частота системной шины: 167 МГц

Объем кэша второго уровня L2: 1024 Кб

Классификатор	**Время обучения**
DecisionTreeClassifier	0,03 сек
KNeighborsClassifier	0,13 сек
RandomForestClassifier	0,7 сек
LogisticRegression	0,09 сек
SVC	4,28 сек
MLPClassifier	0,35 сек
GradientBoosting	0,78 сек

Таблица 2. Алгоритмы классификации и время их обучения

Приложение 2

При написании курсовой работы были задействованы следующие файлы:

- Исследуемый датасет: *'customer churn.csv'*
- Jupyter Notebook, содержащий анализ, очистку данных и обучение алгоритмов с последующей визуализацией: *churn.ipynb',*
- Изображения (в векторном формате): *'ROC_DT.eps', CM_DT.eps', 'ROC_RF.eps', CM_RF.eps', 'ROC_KNN.eps', CM_KNN.eps', 'ROC_LR.eps',*

CM_LR.eps', 'ROC_SVC.eps', CM_SVC.eps', 'ROC_MLP.eps', CM_MLP.eps', 'ROC_GB.eps', CM_GB.eps', 'ROC_GB_mod.eps'.