ДИСЦИПЛИНА **Интеллектуальные системы и технологии**

(полное наименование дисциплины без сокращений)

ИНСТИТУТ **информационных технологий**

КАФЕДРА **корпоративных информационных систем**

полное наименование кафедры

ВИД УЧЕБНОГО МАТЕРИАЛА **Лекция**

(в соответствии с пп.1-11)

ПРЕПОДАВАТЕЛЬ **Демидова Лилия Анатольевна**

(фамилия, имя, отчество)

СЕМЕСТР **1 семестр (осенний), 2024 – 2025 учебный год**

(семестр обучения, учебный год)

ЛЕКЦИЯ 5
СТРАТЕГИИ СЭМПЛИНГА В УСЛОВИЯХ НЕСБАЛАНСИРОВАННОСТИ КЛАССОВ

Нередко возникают ситуации, когда в обучающем наборе данных доля примеров некоторого класса слишком мала. Этот класс будем называть миноритарным, а другой, сильно представленный, – мажоритарным.

Такие ситуации типичны в кредитном скоринге, в медицине, в директ-маркетинге.

Построенный на таких наборах данных классификатор может оказаться абсолютно неэффективным. Могут отличаться и издержки ошибочной классификации.

Обычно неверная классификация примеров миноритарного класса, как правило, обходится в разы дороже, чем ошибочная классификация примера мажоритарного класса.

Одним из подходов для решения проблемы несбалансированности классов является применение различных стратегий сэмплинга, которые можно разделить на две группы: случайные и специальные.

Восстановление баланса классов может проходить двумя путями.

В первом случае удаляют некоторое количество примеров мажоритарного класса (undersampling), во втором – увеличивают количество примеров миноритарного (oversampling).

Уменьшение числа примеров мажоритарного класса

Случайное удаление примеров мажоритарного класса (Random Undersampling)

Это самая простая стратегия балансировки классов.

1. Рассчитывается число K – количество мажоритарных примеров, которое необходимо удалить для достижения требуемого уровня соотношения различных классов.

2. Случайным образом выбираются K мажоритарных примеров и удаляются.

На рисунке 1 изображены примеры некоторого набора данных в двумерном пространстве признаков до и после использования стратегии сэмплинга со случайным удалением примеров мажоритарного класса.

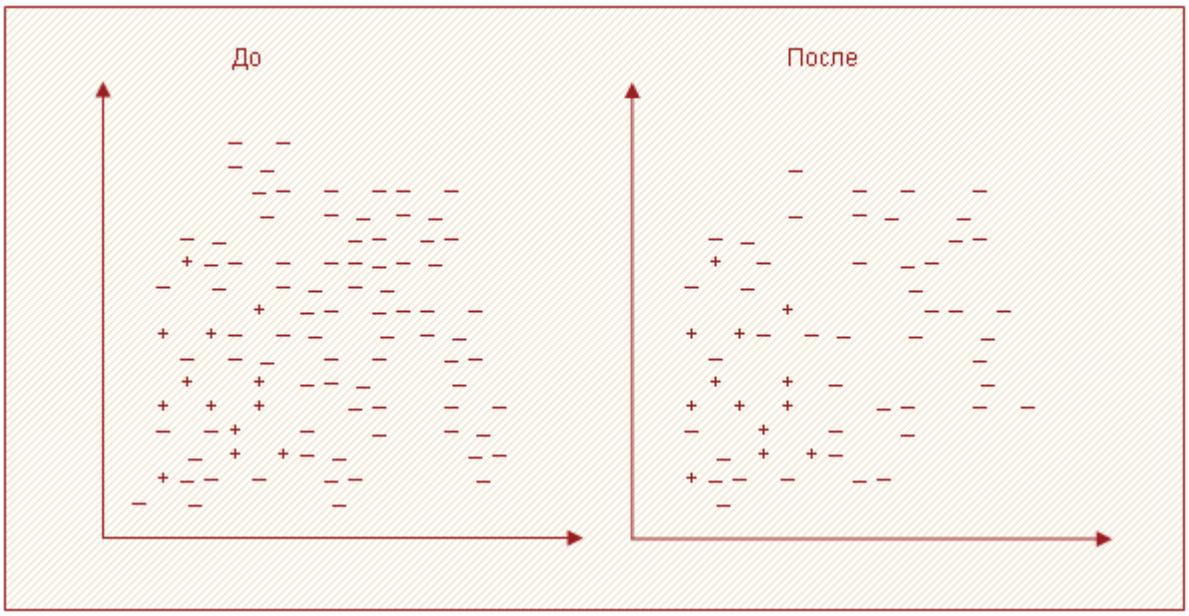

Рисунок 1 – Случайное удаление примеров мажоритарного класса

Примеры из мажоритарного класса могут удаляться не только случайным образом, но и по определенным правилам.

Поиск связей Томека (Tomek Links)

Пусть примеры Ei и Ej принадлежат к различным классам,

$d(Ei,Ej)$ – расстояние между этими примерами.

Пара (Ei,Ej) называется связью Томека, если не найдется ни одного примера El такого, что будет справедлива совокупность неравенств:

$d(Ei,El)<d(Ei,Ej)$;

(1)

$d(Ej,El)<d(Ei,Ej)$.

Все мажоритарные записи, входящие в связи Томека, должны быть удалены из набора данных.

Эта стратегия хорошо удаляет записи, которые можно рассматривать в качестве «зашумляющих». На рисунке 2 визуально показан набор данных в двумерном пространстве признаков до и после применения стратегии поиска связей Томека.

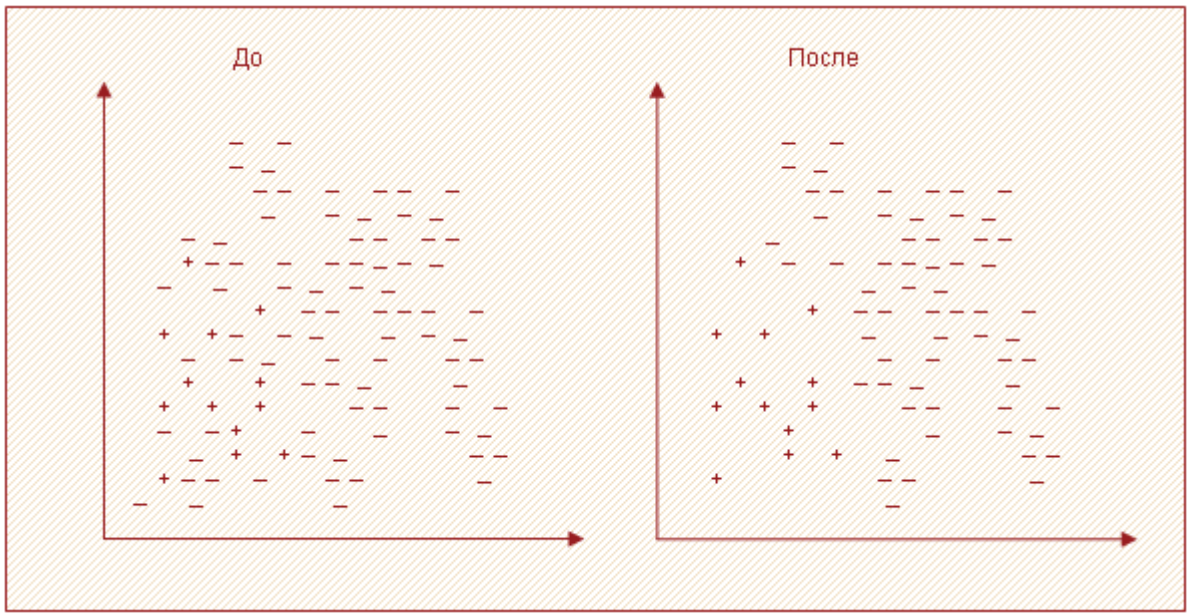

Рисунок 2 – Удаление мажоритарных примеров, участвующих в связях Томека

Правило сосредоточенного ближайшего соседа (Condensed Nearest Neighbor Rule)

Пусть L – исходный набор данных.

Из него выбираются все миноритарные примеры и (случайным образом) один мажоритарный.

Обозначим получившееся множество примеров как *S*.

Все примеры из *L* классифицируются по правилу одного ближайшего соседа (1-NN). Записи, получившие ошибочную метку, добавляются во множество *S* (рисунок 3).

Таким образом, мы будем учить классификатор находить отличие между похожими примерами, но принадлежащими к разным классам.

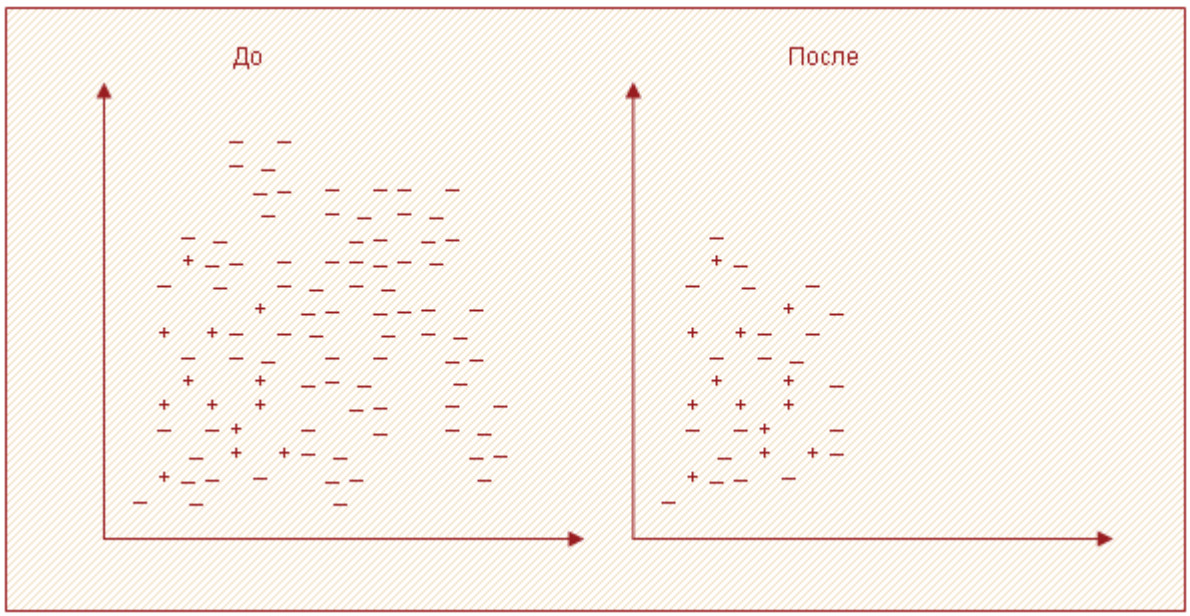

Рисунок 3 – Удаление примеров мажоритарного класса правилом сосредоточенного ближайшего соседа

Односторонний сэмплинг (One-side sampling, one-sided selection)

Главная идея этой стратегии – это последовательное сочетание предыдущих двух, рассмотренных ранее.

На первом шаге применяется правило сосредоточенного ближайшего соседа, а на втором – удаляются все мажоритарные примеры, участвующие в связях Томека.

Таким образом, удаляются большие «сгустки» мажоритарных примеров, а затем область пространства со скоплением миноритарных очищается от потенциальных шумовых эффектов.

Правило «очищающего» соседа (neighborhood cleaning rule)

Эта стратегия также направлена на то, чтобы удалить те примеры, которые негативно влияют на исход классификации миноритарных.

Все примеры классифицируются по правилу трех ближайших соседей.

Удаляются следующие мажоритарные примеры:

– получившие верную метку класса;

– являющиеся соседями миноритарных примеров, которые были неверно классифицированы.

Увеличение числа примеров миноритарного класса

Дублирование примеров миноритарного класса (Oversampling)

Самая простая стратегия – дублирование примеров миноритарного класса.

В зависимости от того, какое соотношение классов необходимо, выбирается количество случайных записей для дублирования.

Такой подход к восстановлению баланса не всегда может оказаться самым эффективным, поэтому был предложен специальный метод увеличения числа примеров миноритарного класса – алгоритм SMOTE (Synthetic Minority Oversampling Technique).

Алгоритм SMOTE (Synthetic Minority Oversampling Technique)

Эта стратегия основана на идее генерации некоторого количества искусственных примеров, которые были бы «похожи» на имеющиеся в миноритарном классе, но при этом не дублировали их.

Для создания нового примера находят разность

$d=Xb–Xa$,

где Xa, Xb – векторы признаков «соседних» примеров a и b из миноритарного класса.

Примеры находят, используя алгоритм ближайшего соседа (или knn-алгоритм).

При этом необходимо и достаточно для примера b получить набор из k соседей, из которого в дальнейшем будет выбрана запись b.

Остальные шаги knn-алгоритма не выполняются.

На основе разности d путем умножения каждого его элемента на случайное число в интервале (0, 1) получают d'.

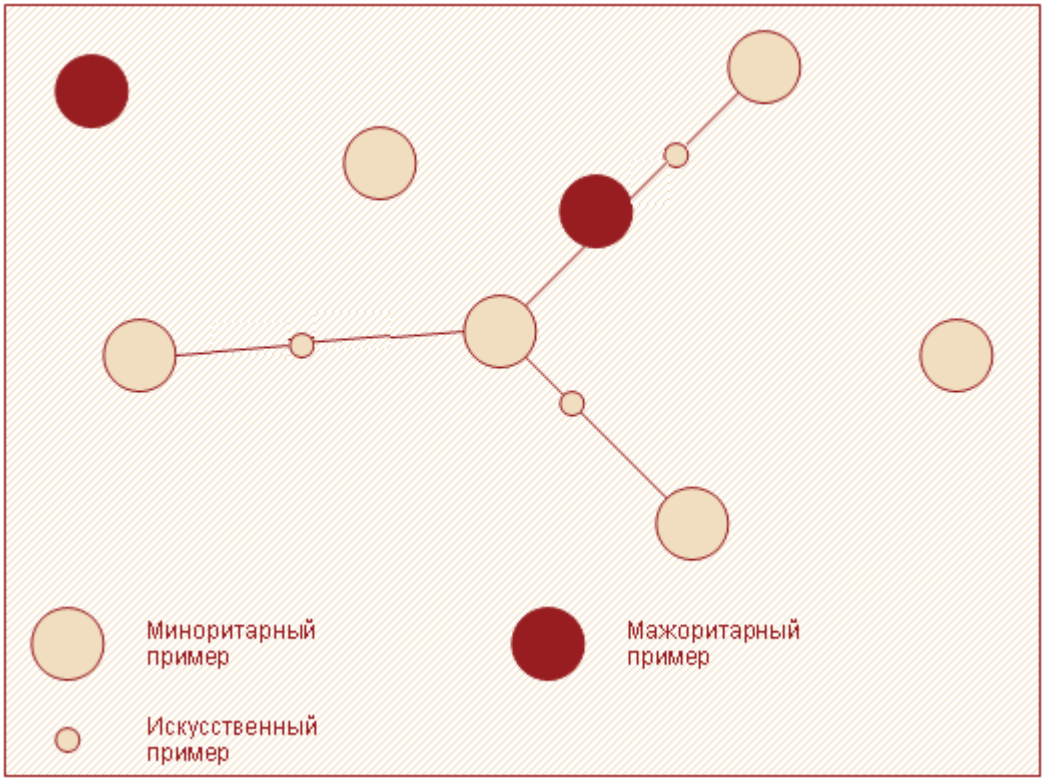

Рисунок 4 – Искусственно созданные новые примеры миноритарного класса

Вектор признаков нового примера вычисляется путем сложения Xa и $d^{\wedge}$.

Алгоритм SMOTE позволяет задавать количество записей, которое необходимо искусственно сгенерировать.

Степень сходства примеров a и b можно регулировать путем изменения значения числа k (числа ближайших соседей).

На рисунке 4 схематично изображено то, как в двумерном пространстве признаков могут располагаться искусственно сгенерированные примеры.

Эта стратегия имеет недостаток в том, что «вслепую» увеличивает плотность примерами в области слабо представленного класса (рисунок 5).

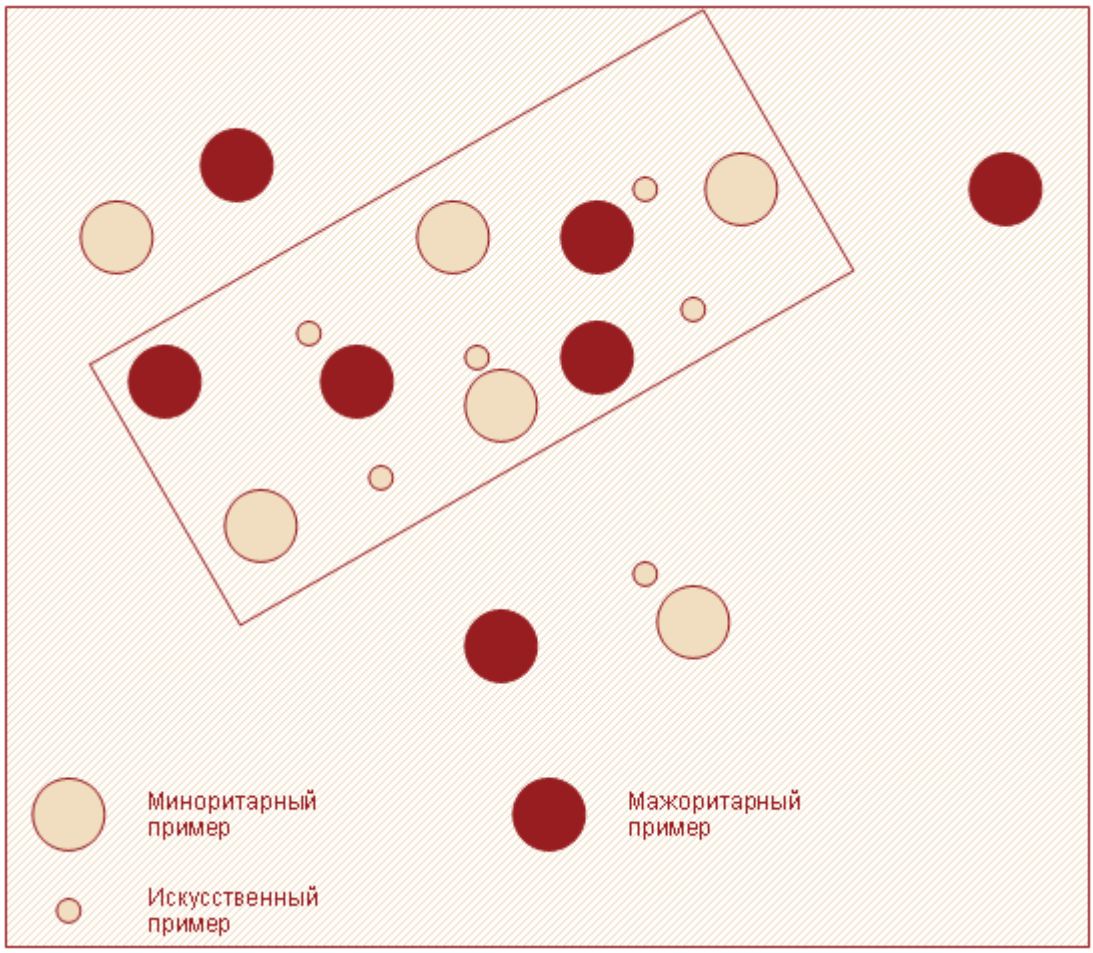

Рисунок 5 – Негативное влияние алгоритма SMOTE

В случае, если миноритарные примеры равномерно распределены среди мажоритарных и имеют низкую плотность, алгоритм SMOTE только сильнее перемешает классы.

В качестве решения данной проблемы был предложен алгоритм адаптивного искусственного увеличения числа примеров миноритарного класса ASMO (Adaptive Synthetic Minority Oversampling).

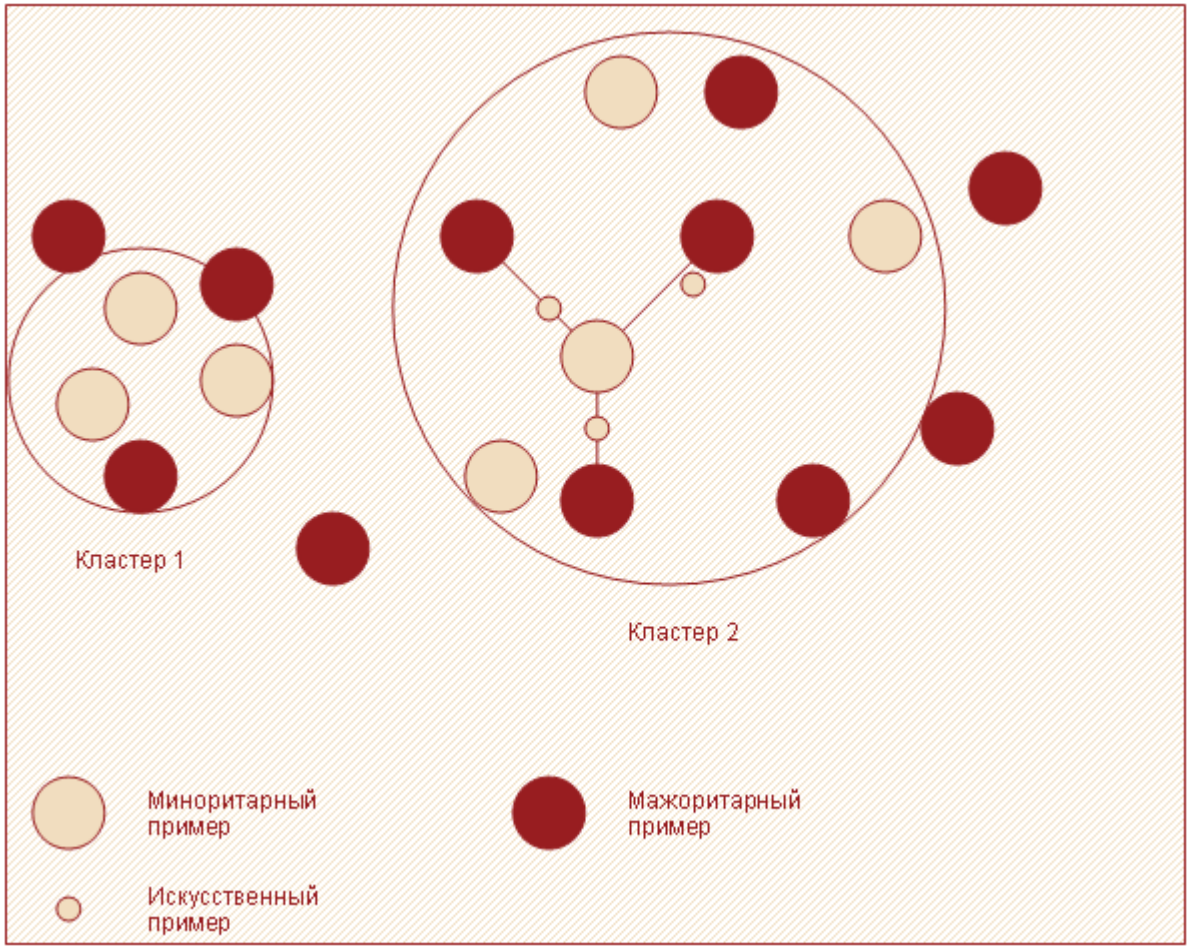

Рисунок 6 – Основная идея алгоритма ASMO

Алгоритм ASMO (Adaptive Synthetic Minority Oversampling)

1. Если для каждого i-го примера миноритарного класса из k ближайших соседей g ($g \leq k$) принадлежит к мажоритарному, то набор данных считается «рассеянным». В этом случае используют алгоритм ASMO, иначе применяют SMOTE (как правило, g задают равным 20).

2. Используя только примеры миноритарного класса, выделить несколько кластеров (например, алгоритмом k-means).

3. Сгенерировать искусственные записи в пределах отдельных кластеров на основе всех классов. Для каждого примера миноритарного класса найти m ближайших соседей и на основе них (также как в SMOTE) создать новые записи.

Такая модификация алгоритма SMOTE делает его более адаптивным к различным наборам данных с несбалансированными классами.

Общее представление идеи данной стратегии показано на рисунке 6.

Пример (задача о моллюсках)

Сравним практически стратегии балансировки на наборе данных о моллюсках (набор данных взят с UCI machine learning repository).

В нем представлены физиологические сведения о моллюсках.

В наборе данных имеются нижеперечисленные поля.

1. Пол.
2. Длина.
3. Диаметр (линия, перпендикулярная длине).
4. Высота.
5. Масса всего моллюска.
6. Масса без раковины.
7. Масса всех внутренних органов (после обескровливания).
8. Масса раковины (после высушивания).

9. Зависимая переменная: количество колец (в год на раковине моллюска появляется 1,5 кольца).

Изначально набор данных был предназначен для решения задачи регрессии. По количеству колец на раковине определяется возраст моллюска. Для классификации в условиях несбалансированности создадим новую выходную переменную, принимающую только два значения. Для этого, предположим, что если количество колец у моллюска не превосходит 18, то для нас он будет считаться молодым, в противном случае – старым.

Проимитируем ситуацию различия издержек и рассмотрим случаи, когда неверное отнесение старого моллюска к молодым может принести бóльшие издержки, чем в случае неверной классификации фактически молодого.

Таким образом, мы получили набор данных с сильно несбалансированными классами, где значение «молодой» было присвоено 4083 записям (97,7%), а значение «старый» – 94 записям (2,3%).

Пусть тестовый и обучающий наборы данных получены стратифицированным сэмплингом.

Стратифицированный семплинг: все записи исходного набора данных разделяются на однородные группы (страты), после чего из каждой группы случайным образом выбираются записи и помещаются в результирующую выборку).

Прежде чем восстанавливать баланс между классами, вернемся к понятию издержек классификации.

Во многих приложениях, таких как кредитный скоринг, директ-маркетинг, издержки при ложноположительной (C10) классификации в несколько раз выше, чем при ложноотрицательной (C01). При пороге отсечения 0,5 количество миноритарных примеров необходимо увеличить в C10/C01 раз (при условии, что C00=C11=0), либо во столько же уменьшить мажоритарный класс.

Сравним следующие стратегии к восстановлению баланса между классами:

– случайное удаление примеров мажоритарного класса;

– дублирование примеров миноритарного класса;

– специальные методы увеличения числа примеров (алгоритмы SMOTE и ASMO).

Для алгоритмов SMOTE и ASMO количество ближайших соседей для генерации примеров установим равным 5.

Алгоритм ASMO признал набор данных нерассеянным (среди 100 ближайших соседей не нашлось даже 20 примеров из мажоритарного класса).

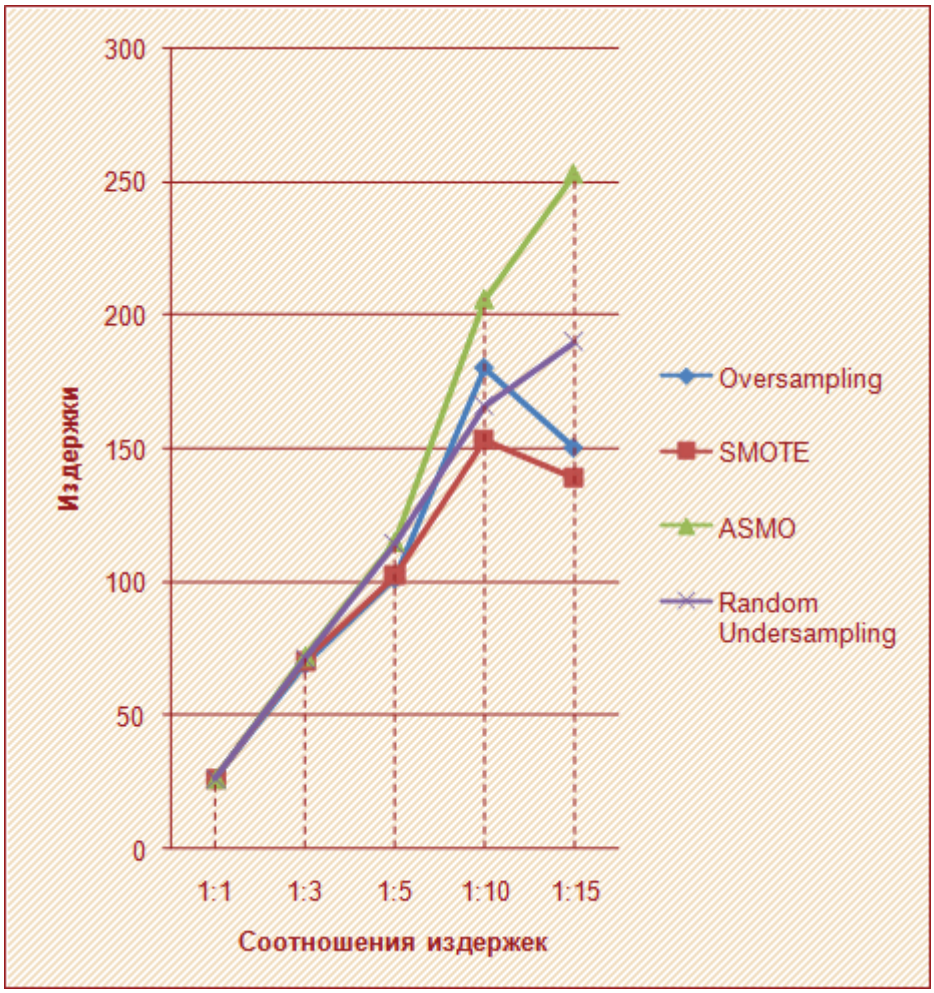

Рисунок 7 – Графики издержек классификации при использовании различных стратегий сэмплинга

Проигнорируем эту информацию и посмотрим, какой будет результат, если генерировать примеры, используя записи из каждого класса.

Для кластеризации будем использовать алгоритм k-means (k = 5).

После восстановления баланса построим логистическую регрессию с порогом отсечения 0,5, и подсчитаем издержки. Результаты представлены на рисунке 7.

Из рисунка 7 видно, что наилучшим образом показал себя алгоритм SMOTE, так как издержки в данном случае оказались самыми меньшими. ASMO проявил себя хуже, однако стоит напомнить, что набор данных не рассеян и согласно данной стратегии необходимо было использовать SMOTE.

Python

imbalanced-learn API

https://imbalanced-learn.readthedocs.io/en/stable/api.html

The `imblearn.under_sampling.prototype_selection` submodule contains methods that select samples in order to balance the dataset.

`under_sampling.CondensedNearestNeighbour` ([...])	Class to perform under-sampling based on the condensed nearest neighbour method.
`under_sampling.EditedNearestNeighbours` ([...])	Class to perform under-sampling based on the edited nearest neighbour method.
`under_sampling.RepeatedEditedNearestNeighbours` ([...])	Class to perform under-sampling based on the repeated edited nearest neighbour method.
`under_sampling.AllKNN` ([sampling_strategy, ...])	Class to perform under-sampling based on the AllKNN method.
`under_sampling.InstanceHardnessThreshold` ([...])	Class to perform under-sampling based on the instance hardness threshold.
`under_sampling.NearMiss` ([sampling_strategy, ...])	Class to perform under-sampling based on NearMiss methods.
`under_sampling.NeighbourhoodCleaningRule` ([...])	Class performing under-sampling based on the neighbourhood cleaning rule.
`under_sampling.OneSidedSelection` ([...])	Class to perform under-sampling based on one-sided selection method.
`under_sampling.RandomUnderSampler` ([...])	Class to perform random under-sampling.
`under_sampling.TomekLinks` ([...])	Class to perform under-sampling by removing Tomek's links.

The `imblearn.over_sampling` provides a set of method to perform over-sampling.

`over_sampling.ADASYN` ([sampling_strategy, ...])	Perform over-sampling using Adaptive Synthetic (ADASYN) sampling approach for imbalanced d
`over_sampling.BorderlineSMOTE` ([...])	Over-sampling using Borderline SMOTE.
`over_sampling.KMeansSMOTE` ([...])	Apply a KMeans clustering before to over-sample using SMOTE.
`over_sampling.RandomOverSampler` ([...])	Class to perform random over-sampling.
`over_sampling.SMOTE` ([sampling_strategy, ...])	Class to perform over-sampling using SMOTE.
`over_sampling.SMOTENC` (categorical_features)	Synthetic Minority Over-sampling Technique for Nominal and Continuous (SMOTE-NC).
`over_sampling.SVMSMOTE` ([sampling_strategy, ...])	Over-sampling using SVM-SMOTE.

The `imblearn.combine` provides methods which combine over-sampling and under-sampling.

`combine.SMOTEENN` ([sampling_strategy, ...])	Class to perform over-sampling using SMOTE and cleaning using ENN.
`combine.SMOTETomek` ([sampling_strategy, ...])	Class to perform over-sampling using SMOTE and cleaning using Tomek links.

The `imblearn.ensemble` module include methods generating under-sampled subsets combined inside an ensemble.

`ensemble.BalanceCascade` (**kwargs)	Create an ensemble of balanced sets by iteratively under-sampling the imbalanced dataset using an
`ensemble.BalancedBaggingClassifier` ([...])	A Bagging classifier with additional balancing.
`ensemble.BalancedRandomForestClassifier` ([...])	A balanced random forest classifier.
`ensemble.EasyEnsemble` (**kwargs)	Create an ensemble sets by iteratively applying random under-sampling.
`ensemble.EasyEnsembleClassifier` ([...])	Bag of balanced boosted learners also known as EasyEnsemble.
`ensemble.RUSBoostClassifier` ([...])	Random under-sampling integrating in the learning of an AdaBoost classifier.

SMOTE

class `imblearn.over_sampling.SMOTE`(*sampling_strategy='auto', random_state=None, k_neighbors=5, m_neighbors='deprecated', out_step='deprecated', kind='deprecated', svm_estimator='deprecated', n_jobs=1, ratio=None*)

ROC-анализ

ROC-кривая (Receiver Operator Characteristic) – кривая, которая наиболее часто используется для представления результатов бинарной классификации в машинном обучении.

Название пришло из систем обработки сигналов.

Поскольку классов два, один из них называется классом с положительными исходами, второй – с отрицательными исходами. ROC-кривая показывает зависимость количества верно классифицированных положительных примеров от количества неверно классифицированных отрицательных примеров. В терминологии ROC-анализа первые называются истинно положительным, вторые – ложно отрицательным множеством.

При этом предполагается, что у классификатора имеется некоторый параметр, варьируя который, мы будем получать то или иное разбиение на два класса.

Этот параметр часто называют порогом, или точкой отсечения (cut-off value). В зависимости от него будут получаться различные величины *ошибок I и II рода*.

В логистической регрессии порог отсечения изменяется от 0 до 1 – это и есть расчетное значение уравнения регрессии. Будем называть его рейтингом.

Для понимания сути ошибок I и II рода рассмотрим четырехпольную таблицу сопряженности (confusion matrix), которая строится на основе результатов классификации моделью и фактической (объективной) принадлежностью примеров к классам.

	Фактически	
Модель	**положительно**	**отрицательно**
положительно	*TP*	*FP*
отрицательно	*FN*	*TN*

В таблице:

— *TP* (*True Positives*) – верно классифицированные положительные примеры (так называемые истинно положительные случаи);

— *TN* (*True Negatives*) – верно классифицированные отрицательные примеры (истинно отрицательные случаи);

— *FN* (*False Negatives*) – положительные примеры, классифицированные как отрицательные (ошибка I рода). Это так называемый «ложный пропуск» – когда интересующее нас событие ошибочно не обнаруживается (ложно отрицательные примеры);

— *FP* (*False Positives*) – отрицательные примеры, классифицированные как положительные (ошибка II рода). Это ложное обнаружение, т.к. при отсутствии события ошибочно выносится решение о его присутствии (ложно положительные случаи).

Что является положительным событием, а что – отрицательным, зависит от конкретной задачи.

Например, если мы прогнозируем вероятность наличия заболевания, то положительным исходом будет класс «Больной пациент», отрицательным – «Здоровый пациент».

И наоборот, если мы хотим определить вероятность того, что человек здоров, то положительным исходом будет класс «Здоровый пациент».

При анализе чаще оперируют не абсолютными показателями, а относительными – долями (rates), выраженными в процентах.

Доля истинно положительных примеров (True Positives Rate):

$TPR=TP/(TP+FN)\cdot 100\%$.

Доля ложно положительных примеров (False Positives Rate):

$FPR=FP/(TN+FP)\cdot 100\%$.

Введем еще два определения: чувствительность и специфичность модели. Ими определяется объективная ценность любого бинарного классификатора.

Чувствительность (Sensitivity) – это и есть доля истинно положительных случаев:

$Se=TPR=TP/(TP+FN)\cdot 100\%$.

Специфичность (Specificity) – доля истинно отрицательных случаев, которые были правильно идентифицированы моделью:

$Sp=TN/(TN+FP)\cdot 100\%$.

При этом $FPR=100 - Sp$.

Модель с высокой чувствительностью часто дает истинный результат при наличии положительного исхода (обнаруживает положительные примеры).

Наоборот, модель с высокой специфичностью чаще дает истинный результат при наличии отрицательного исхода (обнаруживает отрицательные примеры).

Если рассуждать в терминах медицины – задачи диагностики заболевания, где модель классификации пациентов на больных и здоровых называется диагностическим тестом, то получится следующее:

– чувствительный диагностический тест проявляется в гипердиагностике – максимальном предотвращении пропуска больных;

– специфичный диагностический тест диагностирует только доподлинно больных (это важно в случае, когда, например, лечение больного связано с серьезными побочными эффектами и гипердиагностика пациентов не желательна).

ROC-кривая получается следующим образом (рисунок 8):

1. Для каждого значения порога отсечения, которое меняется от 0 до 1 с шагом dx (например, 0.01) рассчитываются значения чувствительности

Se и специфичности *Sp*. В качестве альтернативы порогом может являться каждое последующее значение примера в выборке.

2. Строится график зависимости: по оси Y откладывается чувствительность *Se*, по оси X – 100%–*Sp* (сто процентов минус специфичность, то есть *FPR* – доля ложно положительных случаев).

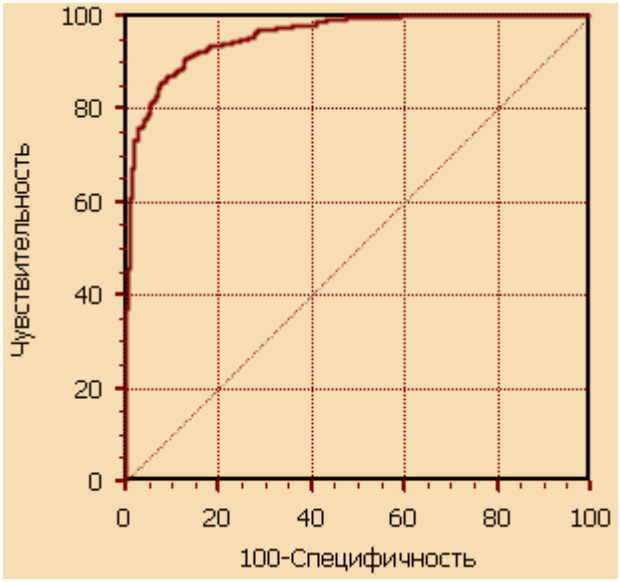

Рисунок 8 – ROC-кривая

График часто дополняют прямой y=x.

Имеется более экономичный способ расчета точек ROC-кривой, чем тот, который приводился выше, т.к. его вычислительная сложность нелинейная и равна $O(n^2)$: для каждого порога необходимо «пробегать» по записям и каждый раз рассчитывать *TP* и *FP*. Если же двигаться вниз по набору данных, отсортированному по убыванию выходного поля классификатора (рейтингу), то можно за один проход вычислить значения всех точек ROC-кривой, последовательно обновляя значения *TP* и *FP*.

Для идеального классификатора график ROC-кривой проходит через верхний левый угол, где доля истинно положительных случаев составляет

100% или 1.0 (идеальная чувствительность), а доля ложно положительных примеров равна нулю. Поэтому чем ближе кривая к верхнему левому углу, тем выше предсказательная способность модели. Наоборот, чем меньше изгиб кривой и чем ближе она расположена к диагональной прямой, тем менее эффективна модель. Диагональная линия соответствует «бесполезному» классификатору, т.е. полной неразличимости двух классов.

При визуальной оценке ROC-кривых расположение их относительно друг друга указывает на их сравнительную эффективность. Кривая, расположенная выше и левее, свидетельствует о большей предсказательной способности модели.

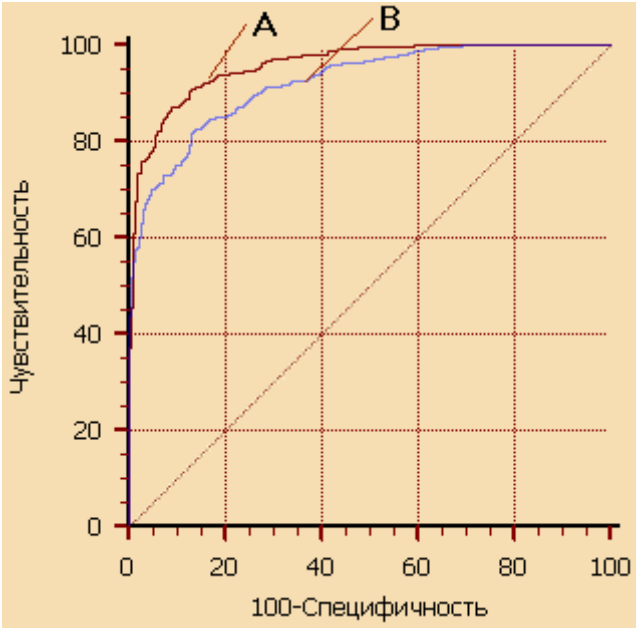

Рисунок 9 – Сравнение ROC-кривых

На рисунке 9 две ROC-кривые совмещены на одном графике. Видно, что модель «А» лучше.

Визуальное сравнение кривых ROC не всегда позволяет выявить наиболее эффективную модель. Своеобразным методом сравнения ROC-кривых является оценка площади под кривыми. Теоретически она изменяется от 0 до 1.0, но, поскольку модель всегда характеризуются кривой, расположенной выше положительной диагонали, то обычно говорят об изменениях от 0.5 («бесполезный» классификатор) до 1.0 («идеальная» модель). Эта

оценка может быть получена непосредственно вычислением площади под многогранником, ограниченным справа и снизу осями координат и слева вверху – экспериментально полученными точками. Численный показатель площади под кривой называется AUC (Area Under Curve). Вычислить его можно, например, с помощью численного метода трапеций.

С большими допущениями можно считать, что чем больше показатель AUC, тем лучшей прогностической силой обладает модель. Однако следует знать, что:

– показатель AUC предназначен скорее для сравнительного анализа нескольких моделей;

– AUC не содержит никакой информации о чувствительности и специфичности модели.

В литературе иногда приводится следующая экспертная шкала для значений AUC, по которой можно судить о качестве модели.

Интервал AUC	**Качество модели**
0.9-1.0	Отличное
0.8-0.9	Очень хорошее
0.7-0.8	Хорошее
0.6-0.7	Среднее
0.5-0.6	Неудовлетворительное

Идеальная модель обладает 100% чувствительностью и специфичностью. Однако на практике добиться этого невозможно, более того, невозможно одновременно повысить и чувствительность, и специфичность модели. Компромисс находится с помощью порога отсечения, т.к. пороговое значение влияет на соотношение *Se* и *Sp*. Можно говорить о задаче нахождения *оптимального порога отсечения* (optimal cut-off value).

Порог отсечения нужен для того, чтобы применять модель на практике: относить новые примеры к одному из двух классов. Для определения оптимального порога нужно задать критерий его определения, т.к. в разных задачах присутствует своя оптимальная стратегия.

Критериями выбора порога отсечения могут выступать следующее.

1. Требование минимальной величины чувствительности (специфичности) модели. Например, нужно обеспечить чувствительность теста не менее 80%. В этом случае оптимальным порогом будет максимальная специфичность (чувствительность), которая достигается при 80% (или значение, близкое к нему «справа» из-за дискретности ряда) чувствительности (специфичности).

2. Требование максимальной суммарной чувствительности и специфичности модели, т.е.

$$Cutt{-}offo = \max_k (Se_k + Sp_k).$$

3. Требование баланса между чувствительностью и специфичностью, т.е. когда $Se \approx Sp$:

$$Cutt{-}offo = \min_k ||Se_k - Sp_k||.$$

Второе значение порога обычно предлагается пользователю по умолчанию. В третьем случае порог есть точка пересечения двух кривых, когда по оси X откладывается порог отсечения, а по оси Y – чувствительность или специфичность модели (рисунок 10).

Второе значение порога обычно предлагается пользователю по умолчанию. В третьем случае порог есть точка пересечения двух кривых, когда по оси X откладывается порог отсечения, а по оси Y – чувствительность или специфичность модели (рисунок 10).

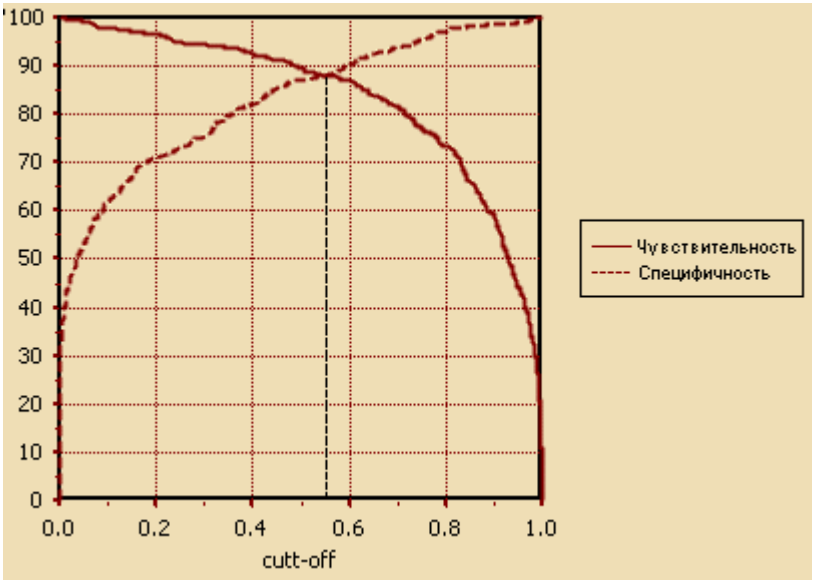

Рисунок 10 – «Точка баланса» между чувствительностью и специфичностью

Существуют и другие подходы, когда ошибкам I и II рода назначается вес, который интерпретируется как цена ошибок. Но здесь встает проблема определения этих весов, что само по себе является сложной, а часто не разрешимой задачей.

Распространенная разновидность научных исследований – разработка и практическое внедрение новых методов диагностики.

Статистическая обработка результатов подобных исследований достаточно специфична.

1. Необходимо сравнить результаты, полученные с использованием новой методики, с результатами применения существующих методов обследования и оценить направление и статистическую значимость имеющихся различий.

2. Необходимо определить значение оцениваемого параметра, при достижении которого будет приниматься решение о наличии патологии – т.н. *точку отсечения* (cut-off value).

3. Необходимо оценить диагностическую значимость новой методики и выразить ее в неких интегральных показателях, допускающих удобное срав-

нение эффективности предлагаемой методики с таковой у других известных методов обследования.

Для решения указанных задач разработан особый метод статистического анализа, называемый «ROC-анализ» (Receiver Operator Characteristic, т.е. «операционная характеристика приёмника», и не спрашивайте у меня, что это значит). ROC-анализ пригоден для описания характеристик диагностических методов, осуществляющих т.н. «бинарную классификацию», т.е. выявляющих наличие некоего состояния (условно обозначаемое «1») либо его отсутствие (условно обозначаемое «0»). К счастью, подобные методы диагностики преобладают в практической медицине, т.к. в основном задача любого диагностического исследования сводится к ответу на вопрос, болен или здоров исследуемый субъект. Соответственно, два варианта выявляемого состояния должны быть *взаимоисключающими* – либо, образно говоря, у пациента есть ВИЧ-инфекция, либо уж ее нет. Такая ситуация, когда наличие некоего исхода однозначно исключает его отсутствие, называется «*исключительное событие*» (exclusive event).

Издержки ошибок классификации (Classification cost error)

(Стоимость ошибок классификации)

Это издержки (потери, убытки) от ошибок классификации, допущенных аналитической моделью.

Такие ошибки могут приводить к неверному принятию решений в бизнесе, а это, в свою очередь, влечет к материальным и финансовым издержкам, которые в аналитических технологиях Data Mining часто называют издержками ошибки классификации.

Например, целью работы модели для оценки кредитоспособности служит выявление добросовестных и недобросовестных клиентов. Данная задача известна как бинарная классификация. Закономерен вопрос: что лучше –

принять добросовестного клиента за недобросовестного (ложно-отрицательная ошибка) или наоборот (ложно-положительная ошибка)?

В первом случае мы теряем только проценты по кредиту, который не был выдан, а во втором – всю сумму, которую получил недобросовестный заемщик, если он не сможет вернуть кредит.

Иными словами, издержки ошибок второго вида больше. Это значит, что при построении модели мы должны минимизировать вероятность появления ошибок, которые приводят к наибольшим потерям.

Типичными примерами, когда издержки неодинаковы для разных типов ошибок, являются:

– кредитный скоринг – издержки выдачи кредита недобросовестным заемщикам существенно выше, чем потери бизнеса из-за отказа в выдаче кредита добросовестным клиентам;

– поиск очагов нефтяного загрязнения – издержки пропуска реального нефтяного пятна существенно выше, чем ложной тревоги;

– техническая диагностика – издержки неправильной идентификации проблемы существенно меньше, чем возможные потери от ее пропуска;

– директ-маркетинг (рассылка прямой почтовой рекламы) – затраты на отправку «макулатурной почты» клиентам, которые не отвечают на нее, существенно меньше, чем потери бизнеса из-за пропуска потенциальных клиентов.

Вообще в экономике и бизнесе трудно найти приложение, в котором издержки различных типов ошибок были одинаковыми, поэтому использование моделей, учитывающих издержки классификации, очень актуально.