# Отчет по заданию №4. Исследование модели Logistic Regression.

Кибитова Валерия 517 группа

МГУ, ВМК, ММП

2015 г.

# Логистическая регрессия

- Несмотря на название – это линейная модель, которая предназначена для для классификации объектов
- В логистической регрессии строится линейный алгоритм классификации:

$$a : X \to Y$$

вида:

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

$< x, w >$ – скалярное произведение признакового описания объекта на вектор весов. Предполагается, что искусственно введён «константный» нулевой признак: $f_0(x) = -1$.

# Логистическая регрессия

- Задача обучения линейного классификатора заключается в том, чтобы по выборке $X^m$ настроить вектор весов w. В логистической регрессии, реализованной в scikit-learn, для этого может быть использована L1 или L2 регуляризация.
- L1 регуляризация:

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

- L2 регуляризация:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

# Логистическая регрессия

- После того, как решение w найдено, становится возможным не только вычислять классификацию для произвольного объекта x, но и оценивать апостериорные вероятности его принадлежности классам:

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

где $\sigma(z) = \frac{1}{1+e^{-z}}$ — сигмоидная функция.

# Параметры логистической регрессии

class sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1)

- solver : 'newton-cg', 'lbfgs', 'liblinear', 'sag'(default = 'liblinear'), – алгоритм, который используется для решения проблемы оптимизации.
- Для небольших данных – 'liblinear', 'sag' – быстрее на больших данных, при этом данные должны быть в одной шкале
- Для многоклассовых задач используются только 'newton-cg' и 'lbfgs'.
- "lbfgs" и "newton-cg" поддреживают только L2 регуляризацию и работают быстрее, чем 'liblinear' для некоторых данных высокой размерности

# Параметры логистической регрессии

- penalty: 'l1', 'l2'(default = 'l2'),– вид регуляризации, "lbfgs" и "newton-cg" поддреживают только L2 регуляризвцию.
- dual: bool(default = False), прямая или дуальная задача оптимизации. Дуальная формулировка поддерживается только с liblinear. Лучше использовать $dual = False$, когда $n\_samples > n\_features$
- C: float(default=1.0)– параметр регуляризации, должен быть положительным float. Меньшее C обеспечивает более сильную регуляризацию.
- fit_intercept : bool(default = True) – определяет добавлять ли константу к решающей функции
- intercept_scaling : float(default = 1) – полезен, только если используется libliner, если self.fit_intercept=True, то вектор x, описывающий объект, становится равным [x, self.intercept_scaling].

# Параметры логистической регрессии

- class_weight : dict or ‘balanced’(default=None)– веса, ассоциированнные с классами, записываются в форме {class_label: weight}. Если параметр не указан, все веса предполагаются равными 1. ’balanced’ использует значения y, чтобы автоматически установить веса обратно пропорционально частотам класса во входных данных.
- max_iter : int(default=100) – используется только для newton-cg, sag и lbfgs. Максимальное число итераций.
- random_state : int seed, RandomState instance, or None (default) – для инициализации генератора псевдослучайных чисел при перемешивании данных.
- multiclass: ’ovr’, ’multinimial’. При использовании ’ovr’ используется стратегия OneVsRest для мультиклассовой классификации. Иначе, минимизируемая функция потерь это multinomial loss, которая учитывает полное распределение вероятностей.

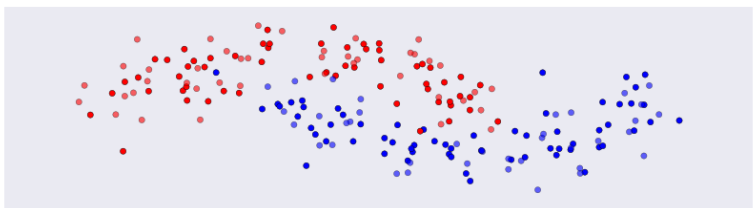
# Эксперименты

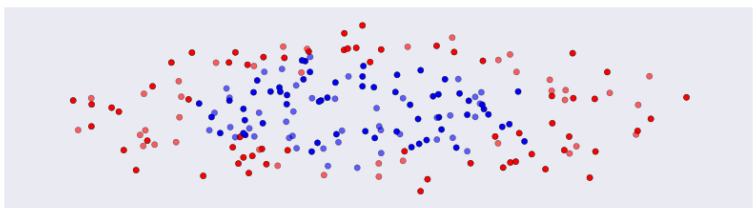
## Поведение классификатора на различных моделях

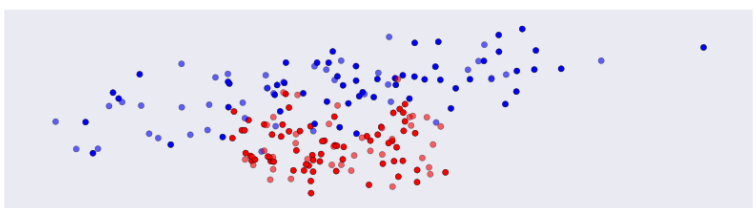
- Модели генирировались с помощью sklearn.datasets:
  make_circles(n_samples=200, noise=0.2, factor=0.5, random_state=1),
  make_moons(n_samples=200, noise=0.2, random_state=0),
  make_classification(n_samples=200, n_features=2, n_redundant=0, n_informative=2, random_state=1, n_clusters_per_class=1)
- В качестве качества использовался roc auc
- Параметр C подбирался с помощью функции LogisticRegressionCV

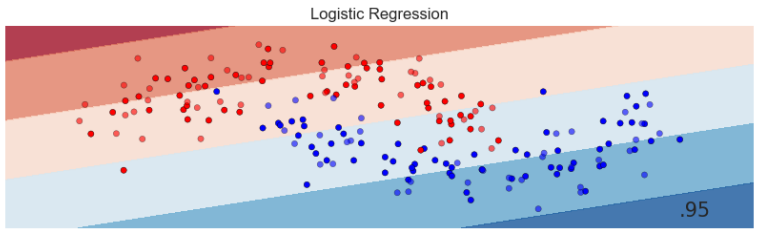
# Эксперименты

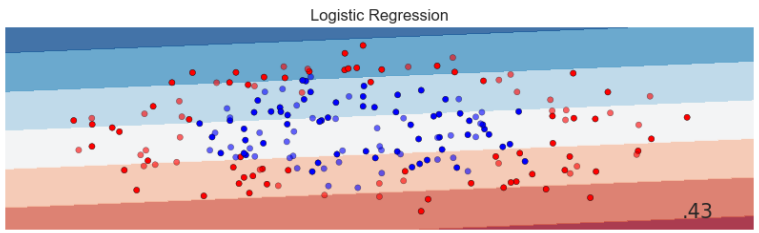
## Поведение классификатора на различных моделях

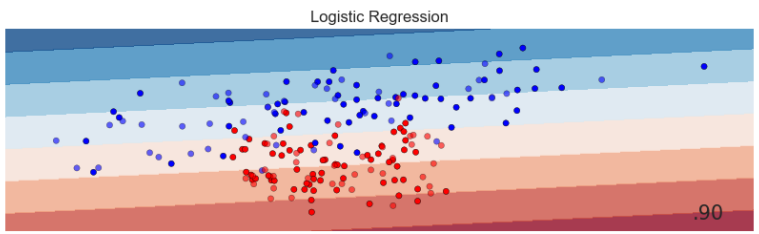






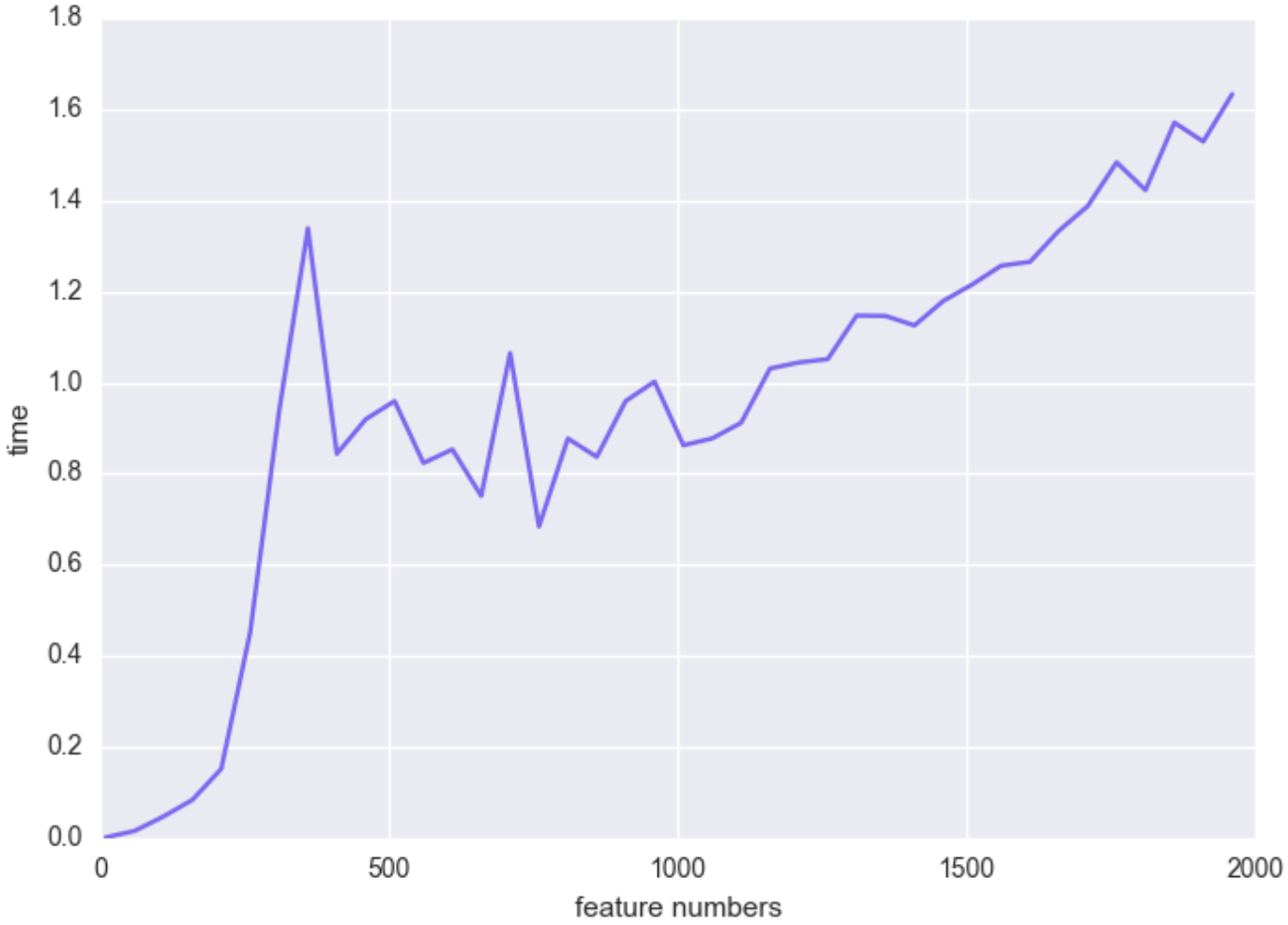


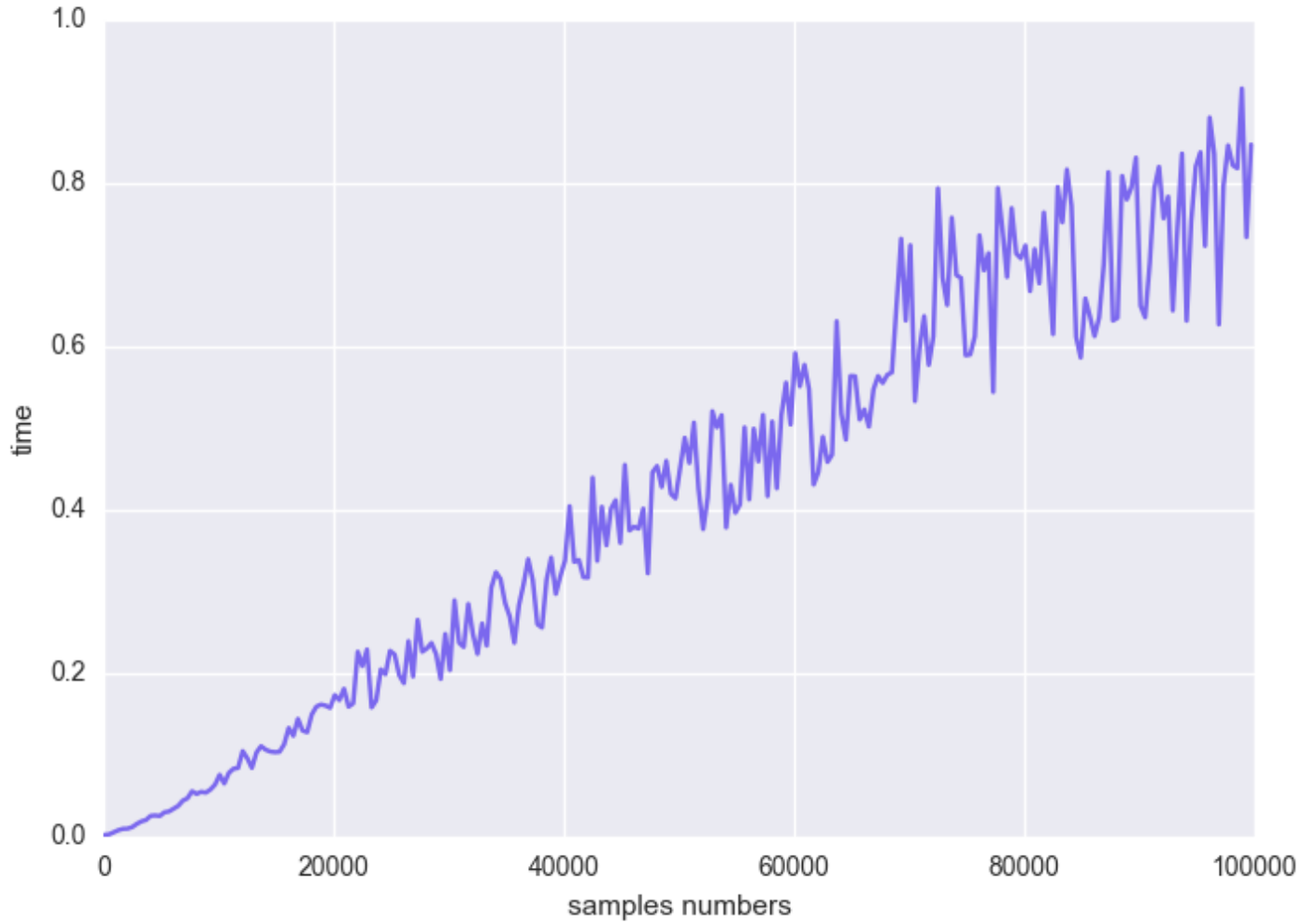
# Эксперименты

## Зависимость скорости классификации от числа признаков



# Эксперименты

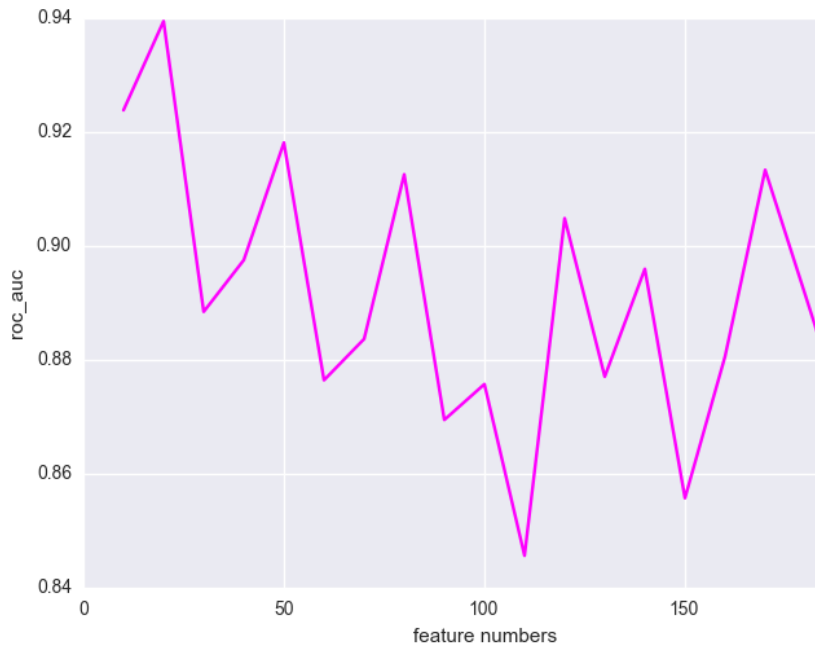
## Зависимость скорости классификации от числа объектов

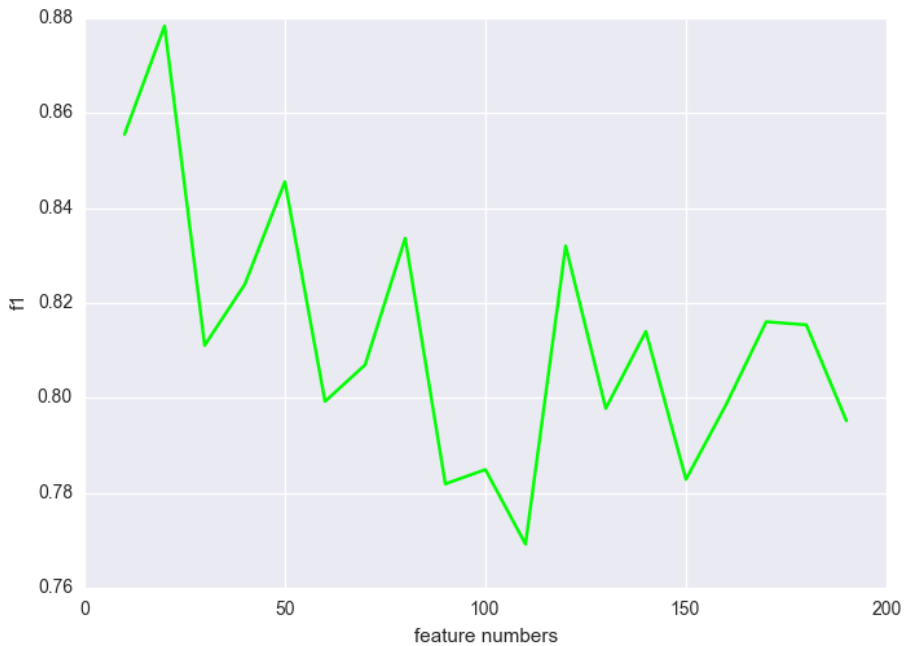


# Эксперименты

## Зависимость качества классификации от числа признаков

Данные генерировались с помощью
sklearn.datasets.make_classification(n_samples=1000, n_features=i, n_informative= i, n_redundant= 0, n_classes=2, n_clusters_per_class=2, weights=None, flip_y=0.01, class_sep=1.0, hypercube=True, shift=0.0, scale=1.0, shuffle=True, random_state=None)

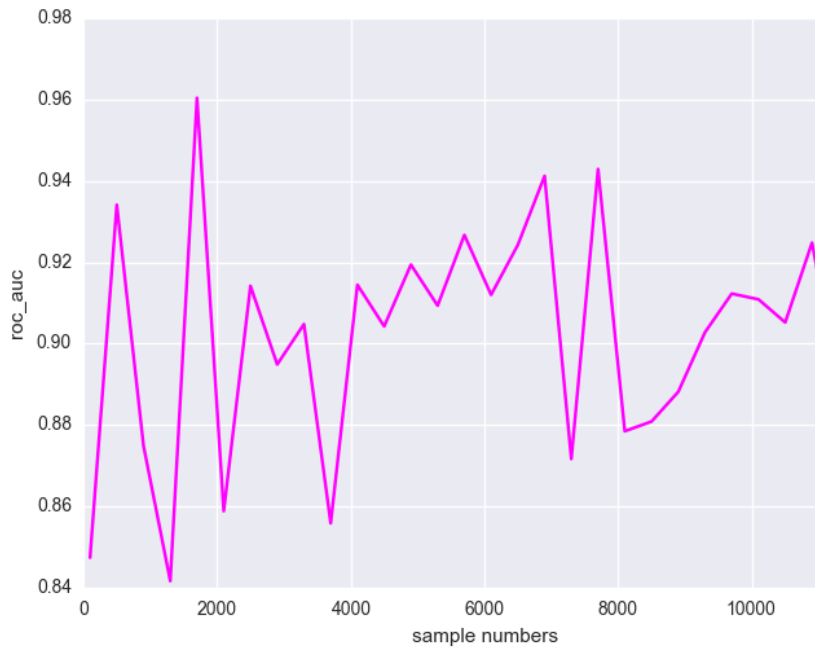


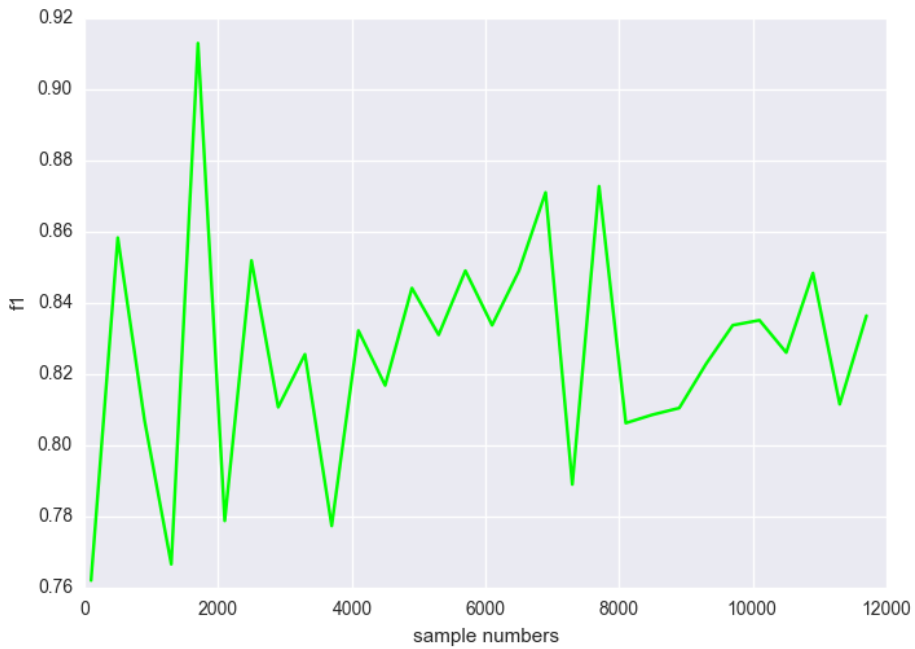



# Эксперименты

## Зависимость качества классификации от числа объектов

Данные генерировались с помощью
sklearn.datasets.make_classification(n_samples=1000, n_features=30, n_informative=30, n_redundant= 0, n_classes=2, n_clusters_per_class=2, weights=None, flip_y=0.01, class_sep=1.0, hypercube=True, shift=0.0, scale=1.0, shuffle=True, random_state=None)

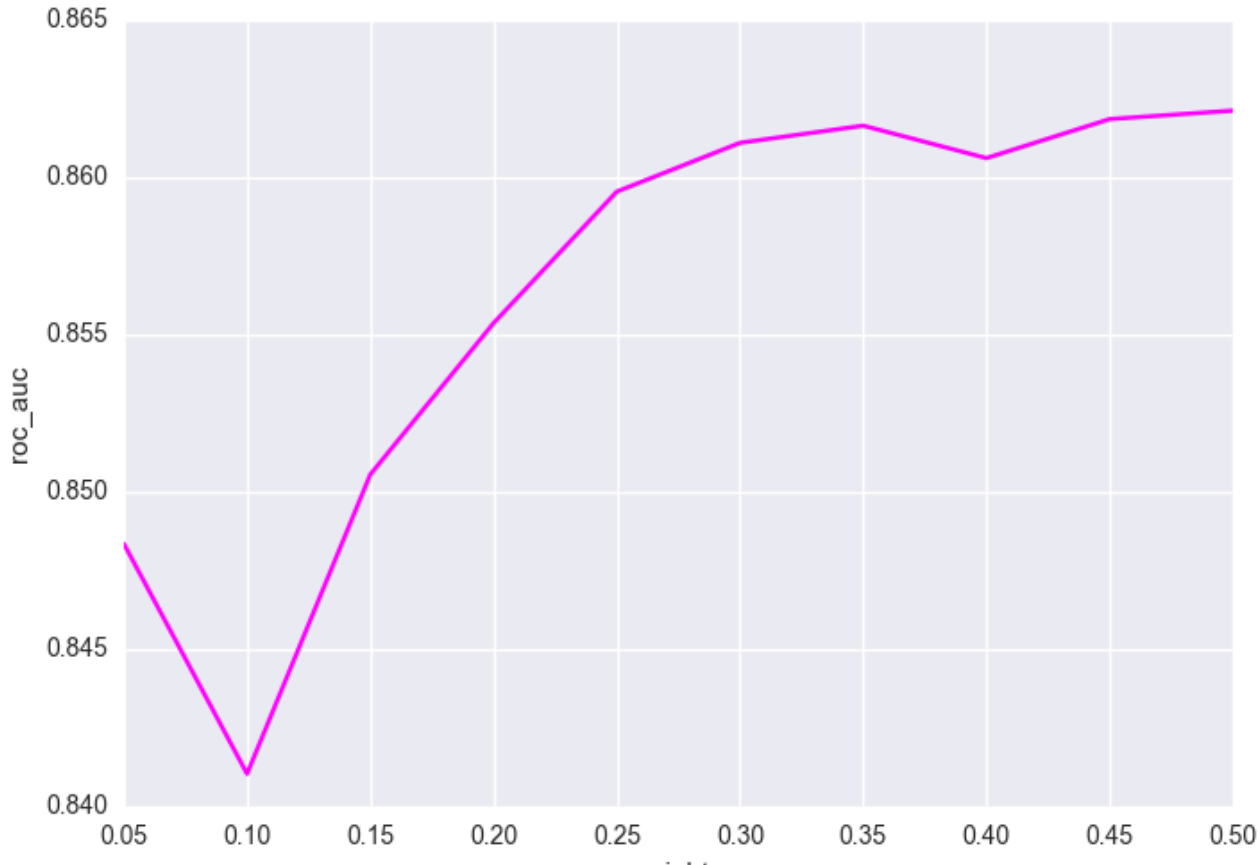


# Эксперименты

## Зависимость качества классификации от сбалансированности классов

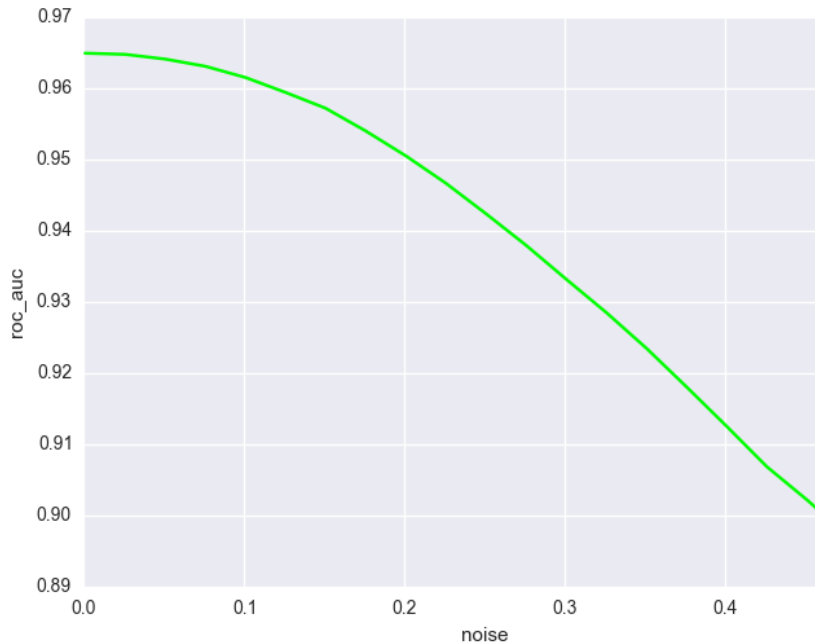
Данные генерировались с помощью
sklearn.datasets.make_classification(n_samples=5000, n_features=30,
n_informative=20, n_redundant= 3, n_classes=2, n_clusters_per_class=2,
weights=[i, 1-i], flip_y=0.01, class_sep=1.0, hypercube=True, shift=0.0, scale=1.0,
shuffle=True, random_state=None)

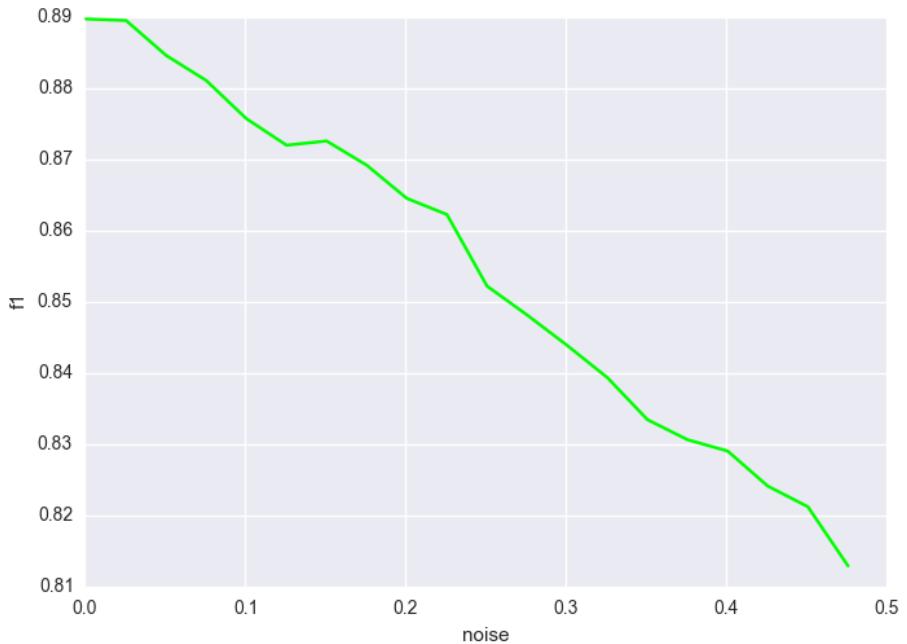


# Эксперименты

## Зависимость качества классификации от шума

Данные генерировались с помощью sklearn.datasets.make_moons(n_samples=1000, shuffle=True, noise=i, random_state=0)



# Выводы

- Алгоритм показывает высокое качество на линейно-разделимых объектах, и низкое на линейно-неразделимых
- Алгоритм работает достаточно быстро на большом количестве объектов
- Качество алгоритма понижается при несбалансированности классов
- Качество алгоритма понижается при добавлении шума

Спасибо за внимание!