Добрый день!

Представляю вашему вниманию вторую статью о соревновании по определению риска дефолта заемщика. В ней я расскажу о разработке дополнительных признаков на основе имеющихся данных, а также продемонстрирую создание простейшей модели машинного обучения.

# Разработка признаков

Соревнования Kaggle с использованием структурированных данных очень часто выигрывают специалисты по разработке признаков: побеждают те, кто может создавать наиболее полезные признаки из данных. Это представляет собой одну из закономерностей в машинном обучении: разработка признаков дает больший возврат инвестиций, чем построение модели и настройка гиперпараметров. Как говорит один из ведущих ученых в области машинного обучения – Эндрю Ын: «Прикладное машинное обучение - это в основном разработка признаков».

Хотя выбор правильной модели и ее оптимальная настройка, несомненно, очень важны, модель может учиться только на основе предоставленных данных. Обеспечение максимального соответствия этих данных задаче – задача специалиста по данным. Конструирование признаков может включать, как создание признаков: добавление новых признаков из существующих данных, так и выбор признаков: выбор только наиболее важных признаков или другие методы уменьшения размерности. Есть много методов, которые мы можем использовать как для создания, так и для выбора признаков.

В рамках этой статьи я рассмотрю два простых метода построения признаков:

* Полиномиальные признаки;
* Признаки, основанные на понимании предметной области.

## Полиномиальные признаки

Одним из простых методов построения признаков является создание [полиномиальных признаков](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html). В этом методе создаются признаки, которые являются степенями существующих признаков, а также определенными взаимодействиями между существующими признаками. Например, мы можем создать переменные EXT\_SOURCE\_1 ^ 2 и EXT\_SOURCE\_2 ^ 2, а также такие переменные, как EXT\_SOURCE\_1 \* EXT\_SOURCE\_2, EXT\_SOURCE\_1 \* EXT\_SOURCE\_2 ^ 2, EXT\_SOURCE\_1 ^ 2 x EXT\_SOURCE\_2 ^ 2 и так далее. Эти признаки, представляющие собой комбинацию нескольких отдельных переменных, называются условиями взаимодействия, поскольку они фиксируют взаимодействия между переменными. Другими словами, хотя две переменные сами по себе могут не иметь сильного влияния на цель, объединение их вместе в одну взаимодействующую переменную может показать связь с целью. Термины взаимодействия обычно используются в статистических моделях, чтобы уловить влияние нескольких переменных, но в машинном обучении используются не очень часто. Тем не менее, мы можем попробовать несколько вариантов, чтобы увидеть, могут ли они помочь нашей модели предсказать, вернет ли клиент ссуду или нет.

Следующим программным кодом я создам полиномиальные признаки, используя переменные EXT\_SOURCE и переменную DAYS\_BIRTH. [В Scikit-Learn есть полезный класс](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html) PolynomialFeatures, который создает полиномы и условия взаимодействия до определенной степени. Достаточно использовать степень 3, чтобы увидеть результат (при создании полиномиальных признаков, нежелательно использовать слишком высокую степень, потому что количество признаков экспоненциально масштабируется со степенью, а также можно столкнуться с [проблемами переобучения](http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html#sphx-glr-auto-examples-model-selection-plot-underfitting-overfitting-py)).

poly\_features = app\_train[['EXT\_SOURCE\_1', 'EXT\_SOURCE\_2', 'EXT\_SOURCE\_3', 'DAYS\_BIRTH', 'TARGET']]

poly\_features\_test = app\_test[['EXT\_SOURCE\_1', 'EXT\_SOURCE\_2', 'EXT\_SOURCE\_3', 'DAYS\_BIRTH']]

from sklearn.preprocessing import Imputer

imputer = Imputer(strategy = 'median')

poly\_target = poly\_features['TARGET']

poly\_features = poly\_features.drop(columns = ['TARGET'])

poly\_features = imputer.fit\_transform(poly\_features)

poly\_features\_test = imputer.transform(poly\_features\_test)

from sklearn.preprocessing import PolynomialFeatures

# Создаем объект PolynomialFeatures, указав степень взаимодействия, равную 3

poly\_transformer = PolynomialFeatures(degree = 3)

poly\_transformer.fit(poly\_features)

poly\_features = poly\_transformer.transform(poly\_features)

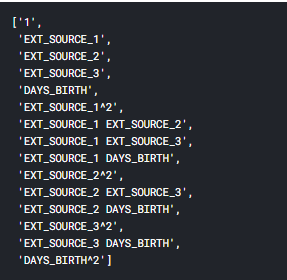
poly\_features\_test = poly\_transformer.transform(poly\_features\_test)

print('Polynomial Features shape: ', poly\_features.shape)

*Polynomial Features shape: (307511, 35)*

Чтобы получить названия новых признаков, нужно использовать метод ***get\_feature\_names***.

poly\_transformer.get\_feature\_names(input\_features = ['EXT\_SOURCE\_1', 'EXT\_SOURCE\_2', 'EXT\_SOURCE\_3', 'DAYS\_BIRTH'])[:15]



Теперь у нас имеется 35 признаков новых: индивидуальные признаки; признаки, повышенные до степени 3; взаимодействующие признаки. Можно посмотреть, коррелируют ли какие-либо из этих новых признаков с целевой меткой.

poly\_features = pd.DataFrame(poly\_features,

columns = poly\_transformer.get\_feature\_names(['EXT\_SOURCE\_1', 'EXT\_SOURCE\_2',

'EXT\_SOURCE\_3', 'DAYS\_BIRTH']))

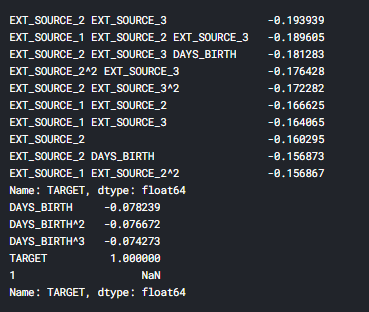
poly\_features['TARGET'] = poly\_target

# Найдем корреляцию с целевой меткой

poly\_corrs = poly\_features.corr()['TARGET'].sort\_values()

print(poly\_corrs.head(10))

print(poly\_corrs.tail(5))



Некоторые из новых признаков имеют большую (с точки зрения абсолютной величины) корреляцию с целью, чем исходные признаки. При создании модели машинного обучения, можно попробовать использовать или не использовать эти признаки, чтобы определить, действительно ли они помогают модели учиться.

Добавим эти признаки в копию данных обучения и тестирования, а затем оценим модели с этими признаками и без них. Часто в машинном обучении единственный способ узнать, сработает ли тот или иной подход, - это попробовать его.

poly\_features\_test = pd.DataFrame(poly\_features\_test,

columns = poly\_transformer.get\_feature\_names(['EXT\_SOURCE\_1', 'EXT\_SOURCE\_2',

'EXT\_SOURCE\_3', 'DAYS\_BIRTH']))

poly\_features['SK\_ID\_CURR'] = app\_train['SK\_ID\_CURR']

app\_train\_poly = app\_train.merge(poly\_features, on = 'SK\_ID\_CURR', how = 'left')

poly\_features\_test['SK\_ID\_CURR'] = app\_test['SK\_ID\_CURR']

app\_test\_poly = app\_test.merge(poly\_features\_test, on = 'SK\_ID\_CURR', how = 'left')

app\_train\_poly, app\_test\_poly = app\_train\_poly.align(app\_test\_poly, join = 'inner', axis = 1)

print('Training data with polynomial features shape: ', app\_train\_poly.shape)

print('Testing data with polynomial features shape: ', app\_test\_poly.shape)

*Training data with polynomial features shape: (307511, 275)*

*Testing data with polynomial features shape: (48744, 275)*

## Признаки, основанные на понимании предметной области

Теперь попробуем сконструировать несколько признаков, которые попытаются уловить то, что, возможно, может быть важным для определения того, не совершит ли клиент дефолт по ссуде. Для знакомства с конструированием подобных признаков я использую информацию публичного [скрипта](https://www.kaggle.com/jsaguiar/updated-0-792-lb-lightgbm-with-simple-features):

* CREDIT\_INCOME\_PERCENT: процент суммы кредита относительно дохода клиента;
* ANNUITY\_INCOME\_PERCENT: процент аннуитетного платежа кредита относительно дохода клиента;
* CREDIT\_TERM: продолжительность платежа в месяцах;
* DAYS\_EMPLOYED\_PERCENT: процент отработанных дней по отношению к возрасту клиента.

app\_train\_domain = app\_train.copy()

app\_test\_domain = app\_test.copy()

app\_train\_domain['CREDIT\_INCOME\_PERCENT'] = app\_train\_domain['AMT\_CREDIT'] / app\_train\_domain['AMT\_INCOME\_TOTAL']

app\_train\_domain['ANNUITY\_INCOME\_PERCENT'] = app\_train\_domain['AMT\_ANNUITY'] / app\_train\_domain['AMT\_INCOME\_TOTAL']

app\_train\_domain['CREDIT\_TERM'] = app\_train\_domain['AMT\_ANNUITY'] / app\_train\_domain['AMT\_CREDIT']

app\_train\_domain['DAYS\_EMPLOYED\_PERCENT'] = app\_train\_domain['DAYS\_EMPLOYED'] / app\_train\_domain['DAYS\_BIRTH']

app\_test\_domain['CREDIT\_INCOME\_PERCENT'] = app\_test\_domain['AMT\_CREDIT'] / app\_test\_domain['AMT\_INCOME\_TOTAL']

app\_test\_domain['ANNUITY\_INCOME\_PERCENT'] = app\_test\_domain['AMT\_ANNUITY'] / app\_test\_domain['AMT\_INCOME\_TOTAL']

app\_test\_domain['CREDIT\_TERM'] = app\_test\_domain['AMT\_ANNUITY'] / app\_test\_domain['AMT\_CREDIT']

app\_test\_domain['DAYS\_EMPLOYED\_PERCENT'] = app\_test\_domain['DAYS\_EMPLOYED'] / app\_test\_domain['DAYS\_BIRTH']

## Визуализируем новые признаки

Необходимо визуально изучить эти признаки, отражающие знания о предметной области. Для всего этого построим график KDE, окрашенный значением TARGET.

plt.figure(figsize = (12, 20))

for i, feature in enumerate(['CREDIT\_INCOME\_PERCENT', 'ANNUITY\_INCOME\_PERCENT', 'CREDIT\_TERM', 'DAYS\_EMPLOYED\_PERCENT']):

plt.subplot(4, 1, i + 1)

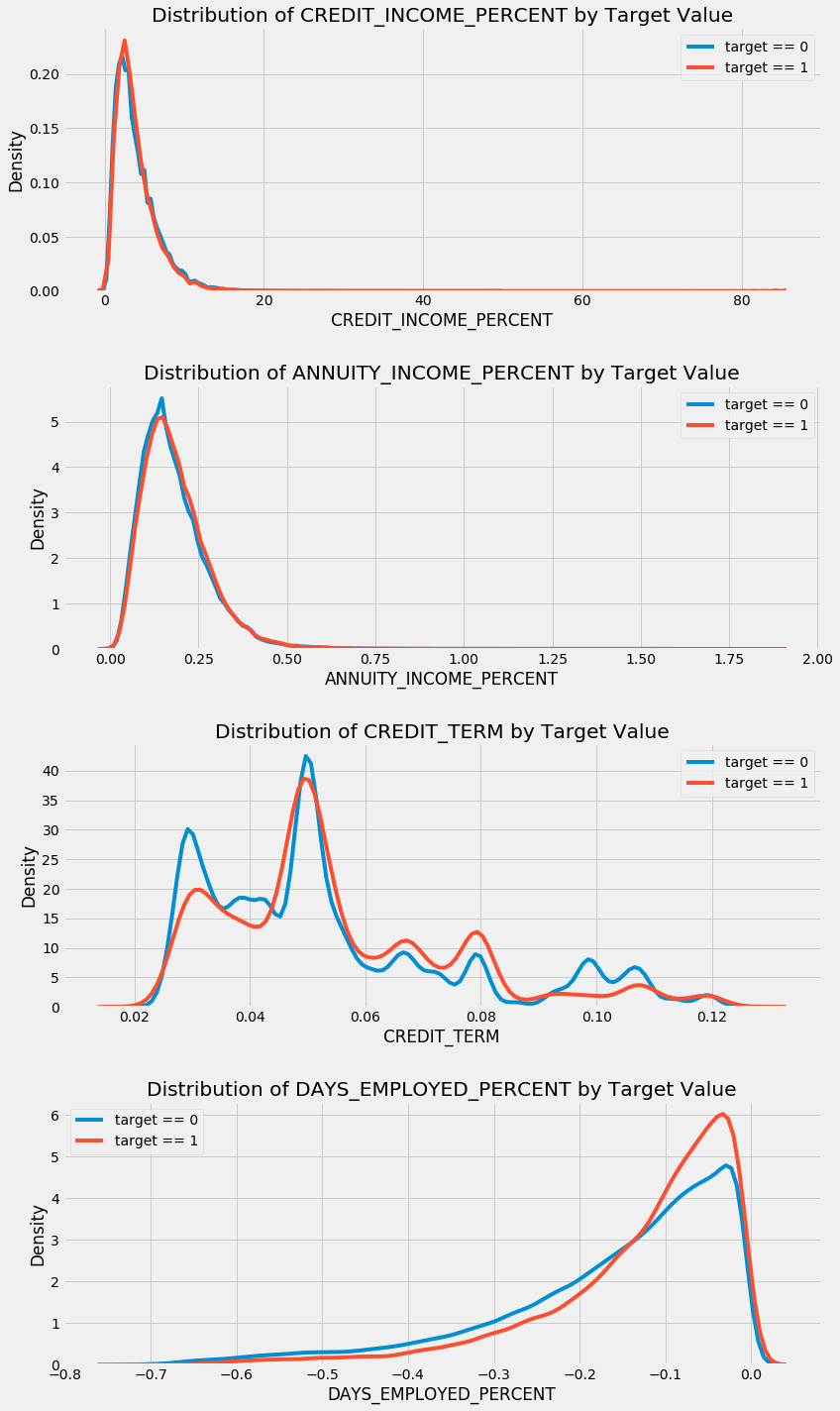
sns.kdeplot(app\_train\_domain.loc[app\_train\_domain['TARGET'] == 0, feature], label = 'target == 0')

sns.kdeplot(app\_train\_domain.loc[app\_train\_domain['TARGET'] == 1, feature], label = 'target == 1')

plt.title('Distribution of %s by Target Value' % feature)

plt.xlabel('%s' % feature); plt.ylabel('Density');

plt.tight\_layout(h\_pad = 2.5)



Сложно сказать заранее, будут ли полезны эти новые признаки. Единственный способ узнать наверняка - попробовать их.

# Baseline

Для наивного решения можно было бы указать одно и то же значение для всех примеров из тестового набора. Нас просят предсказать вероятность невыплаты ссуды, поэтому можно предположить вероятность 0,5 для всех наблюдений на тестовой выборке. Это даст нам площадь под кривой ошибок (AUC ROC), равную 0,5. Но лучше воспользоваться для нашего бейзлайна чуть более сложной моделью: логистической регрессией.

## Реализация логистической регрессии

Чтобы получить базовый уровень, я использую рассмотренные ранее признаки, за исключением созданных вручную. Предварительно обработаю данные, заполнив недостающие значения и нормализуя диапазон признаков (масштабирование признаков). Следующий код выполняет оба этих шага предварительной обработки.

from sklearn.preprocessing import MinMaxScaler, Imputer

if 'TARGET' in app\_train:

train = app\_train.drop(columns = ['TARGET'])

else:

train = app\_train.copy()

features = list(train.columns)

test = app\_test.copy()

# Заполняем пропущенные значения медианным значением признака

imputer = Imputer(strategy = 'median')

# Масштабируем признаки в диапазон 0-1

scaler = MinMaxScaler(feature\_range = (0, 1))

imputer.fit(train)

train = imputer.transform(train)

test = imputer.transform(app\_test)

scaler.fit(train)

train = scaler.transform(train)

test = scaler.transform(test)

print('Training data shape: ', train.shape)

print('Testing data shape: ', test.shape)

*Training data shape: (307511, 240)*

*Testing data shape: (48744, 240)*

Для первой модели я использую LogisticRegression [из библиотеки Scikit-Learn](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html). Единственное изменение, которое потребуется в настройки модели по умолчанию, - уменьшу [параметр регуляризации](http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression) **C**, контролирующий степень переобучения (более низкое значение должно уменьшить переобучение). Это даст чуть лучшие результаты, чем LogisticRegression с параметрами по умолчанию, но все равно установит низкую планку для любых будущих моделей.

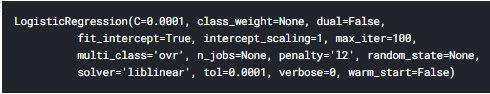
Сначала создаем модель, затем обучаем ее с помощью метода **.fit**, а затем делаем прогнозы на основе тестовых данных с помощью метода **.predict\_proba** (нам нужны вероятности, а не целые числа 0 или 1).

from sklearn.linear\_model import LogisticRegression

# Создаем модель и тренируем ее

log\_reg = LogisticRegression(C = 0.0001)

log\_reg.fit(train, train\_labels)



Теперь, когда модель обучена, можно использовать ее для прогнозирования. В задаче требуется спрогнозировать вероятность невыплаты ссуды, поэтому использую метод модели **predict.proba**. Он возвращает массив m\*2, где m - количество наблюдений. Первый столбец - это вероятность того, что цель будет равна 0, а второй – вероятность того, что цель будет равна 1 (поэтому для каждой строки два столбца в сумме должны быть равными 1). Нам нужна вероятность того, что ссуда не будет погашена, поэтому выберу второй столбец.

Следующая строка кода формирует прогнозы и выбирает правильный столбец.

log\_reg\_pred = log\_reg.predict\_proba(test)[:, 1]

Прогнозы должны быть в формате, показанном в файле sample\_submission.csv, где есть только два столбца: SK\_ID\_CURR и TARGET. Создадим фрейм данных в этом формате из набора тестов и прогнозов.

submit = app\_test[['SK\_ID\_CURR']]

submit['TARGET'] = log\_reg\_pred

submit.head()



Прогнозы представляют собой вероятность от 0 до 1 того, что ссуда не будет возвращена. Если бы требовалось использовать их для классификации заемщиков, можно было бы установить порог вероятности для определения рискованности ссуды.

submit.to\_csv('log\_reg\_baseline.csv', index = False)

Файл с результатом теперь сохранен в виртуальной среде, в которой работает наш код. Чтобы получить доступ к нему, в конце программы необходимо нажать синюю кнопку «Зафиксировать и запустить» в правом верхнем углу экрана. Это действие запускает весь код, а затем позволяет загружать любые файлы, созданные во время запуска.

После запуска программы созданные файлы будут доступны на вкладке «Версии» раздела «Вывод». Отсюда файлы с результатами можно отправить на оценку или скачать. Поскольку в этой программе несколько моделей, файлов результата будет несколько.

При отправке результата на оценку базовый показатель данной модели должен быть приблизительно равен 0,671.

## Улучшенная модель: случайный лес

Чтобы попытаться улучшить низкую производительность базовой модели, можно обновить алгоритм. Давайте попробуем использовать случайный лес на тех же данных обучения, чтобы увидеть, как это влияет на производительность. Случайный лес - гораздо более мощная модель, особенно при использовании сотен деревьев.

from sklearn.ensemble import RandomForestClassifier

random\_forest = RandomForestClassifier(n\_estimators = 100, random\_state = 50, verbose = 1, n\_jobs = -1)

random\_forest.fit(train, train\_labels)

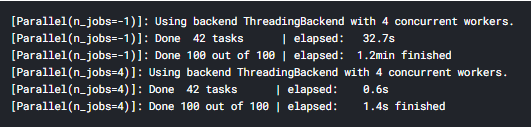
# Определяем важность признаков

feature\_importance\_values = random\_forest.feature\_importances\_

feature\_importances = pd.DataFrame({'feature': features, 'importance': feature\_importance\_values})

# Формируем предсказания для тестовых данных

predictions = random\_forest.predict\_proba(test)[:, 1]



Оценка этой модели чуть выше базового уровня и составляет примерно 0,678 балла.

## Делаем прогнозы, используя специальные признаки

Единственный способ увидеть, улучшили ли модель полиномиальные признаки и признаки на основе знания предметной области, - это обучить модель с их использованием. Затем можно сравнить полученную оценку с оценкой для модели без этих признаков, чтобы оценить эффект от разработки признаков.

poly\_features\_names = list(app\_train\_poly.columns)

imputer = Imputer(strategy = 'median')

poly\_features = imputer.fit\_transform(app\_train\_poly)

poly\_features\_test = imputer.transform(app\_test\_poly)

scaler = MinMaxScaler(feature\_range = (0, 1))

poly\_features = scaler.fit\_transform(poly\_features)

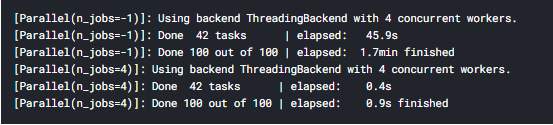
poly\_features\_test = scaler.transform(poly\_features\_test)

random\_forest\_poly = RandomForestClassifier(n\_estimators = 100, random\_state = 50, verbose = 1, n\_jobs = -1)

random\_forest\_poly.fit(poly\_features, train\_labels)

# Формируем предсказания на тестовых данных

predictions = random\_forest\_poly.predict\_proba(poly\_features\_test)[:, 1]



Эта модель так же, как и модель без специальных признаков, набирает 0,678 балла. Учитывая эти результаты, не похоже, что полиномиальные признаки помогли в этом случае.

## Тестирование признаков, полученных с помощью знания предметной области

Теперь протестируем и другие созданные нами признаки.

app\_train\_domain = app\_train\_domain.drop(columns = 'TARGET')

domain\_features\_names = list(app\_train\_domain.columns)

imputer = Imputer(strategy = 'median')

domain\_features = imputer.fit\_transform(app\_train\_domain)

domain\_features\_test = imputer.transform(app\_test\_domain)

scaler = MinMaxScaler(feature\_range = (0, 1))

domain\_features = scaler.fit\_transform(domain\_features)

domain\_features\_test = scaler.transform(domain\_features\_test)

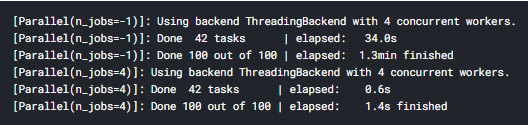
random\_forest\_domain = RandomForestClassifier(n\_estimators = 100, random\_state = 50, verbose = 1, n\_jobs = -1)

random\_forest\_domain.fit(domain\_features, train\_labels)

feature\_importance\_values\_domain = random\_forest\_domain.feature\_importances\_

feature\_importances\_domain = pd.DataFrame({'feature': domain\_features\_names, 'importance': feature\_importance\_values\_domain})

predictions = random\_forest\_domain.predict\_proba(domain\_features\_test)[:, 1]



Оценка для этой модели составляет 0,679 балла, что указывает на низкую значимость разработанных признаков в этом случае.

## Интерпретация модели: важность признаков

В качестве простого метода определения наиболее релевантных признаков можно взглянуть на важность признаков для модели случайного леса. Учитывая корреляции, которые мы видели в исследовательском анализе данных, следует ожидать, что наиболее важными признаками являются EXT\_SOURCE и DAYS\_BIRTH.

def plot\_feature\_importances(df):

df = df.sort\_values('importance', ascending = False).reset\_index()

df['importance\_normalized'] = df['importance'] / df['importance'].sum()

plt.figure(figsize = (10, 6))

ax = plt.subplot()

ax.barh(list(reversed(list(df.index[:15]))),

df['importance\_normalized'].head(15),

align = 'center', edgecolor = 'k')

ax.set\_yticks(list(reversed(list(df.index[:15]))))

ax.set\_yticklabels(df['feature'].head(15))

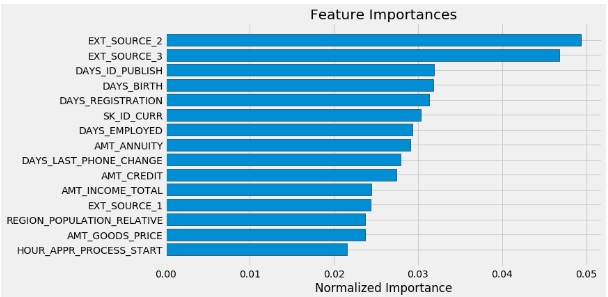
plt.xlabel('Normalized Importance'); plt.title('Feature Importances')

plt.show()

return df

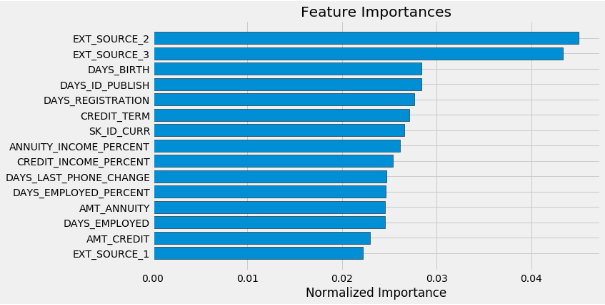
# Отображаем важность признаков без учета специальных признаков

feature\_importances\_sorted = plot\_feature\_importances(feature\_importances)



Как и ожидалось, наиболее важные признаки связаны с EXT\_SOURCE и DAYS\_BIRTH. Можно заметить, что существует лишь несколько признаков, имеющих существенное значение для модели. Это говорит о том, что можно отказаться от многих признаков без снижения качества (и даже увидеть его увеличение). Это не самый изощренный метод интерпретации модели или уменьшения размерности, но он позволяет начать понимать, какие факторы модель принимает во внимание при прогнозировании.

feature\_importances\_domain\_sorted = plot\_feature\_importances(feature\_importances\_domain)



Все четыре созданные вручную признака вошли в топ-15 самых важных. Это должно вселить в нас уверенность в том, что наши знания в предметной области хотя бы частично соответствуют правильному пути.

## Выводы

В этой и предыдущей статьях я постарался показать, как начать соревнования по машинному обучению Kaggle. Сначала постарайтесь понять данные, задачу и метрику, по которой будут оцениваться результаты. Затем проведите простой EDA, чтобы попытаться определить взаимосвязи, тенденции или аномалии, которые могут помочь в моделировании. Попутно можно выполнить необходимые шаги предварительной обработки, такие как кодирование категориальных переменных, заполнение пропущенных значений и масштабирование признаков. Затем создайте новые признаки на основе существующих данных, чтобы посмотреть, может ли это помочь модели.

По завершении исследования данных, подготовки данных и проектирования признаков следует реализовать базовую модель. Затем постройте немного более сложную модель, чтобы побить первый результат. И так далее. Кроме того, необходимо определить эффект добавления инженерных признаков.

В целом, необходимо следовать общей схеме [проекта машинного обучения](https://towardsdatascience.com/a-complete-machine-learning-walk-through-in-python-part-one-c62152f39420):

1. Разберитесь в проблеме и данных
2. Очистите и отформатируйте данные (в основном это сделали для нас организаторы конкурса)
3. Проведите исследовательский анализ данных
4. Создайте базовую модель
5. Улучшить модель
6. Интерпретируйте модель

Соревнования по машинному обучению немного отличаются от типичных задач науки о данных тем, что в их рамках участники озабочены только достижением наилучших результатов по одному показателю и не заботятся об интерпретации. Однако, пытаясь понять, как модели принимают решения, мы можем попытаться улучшить их или изучить ошибки, чтобы исправить их.

Я надеюсь, что моя статья помогла вам принять участие в этом (или каком-либо другом) соревновании по машинному обучению, и что теперь вы готовы самостоятельно и с помощью сообщества начать дальнейшую работу над серьезными проблемами!

При подготовке статьи были использованы материалы из открытых источников: [источник\_1](https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction), [источник\_2](https://www.kaggle.com/codename007/home-credit-complete-eda-feature-importance).