Санкт-Петербургский Государственный Университет

Курсовая работа на тему
“Предсказание вероятности дефолта по кредиту”

Студент 231 группы:
Алтынова Анна Юрьевна

Научный руководитель:
Григорьев Д.А.

27 декабря 2019 г.

Введение

Для финансовых организаций крайне важно измерение финансового риска, то есть вероятности потерь финансовых ресурсов компании. Одним из видов финансовго риска является кредитный риск — вероятность неисполнения заемщиком своих обязательств, то есть вероятность возникновения дефолта. Дефолт, объявленный физическим лицом - это невыплата кредитных платежей.

Предсказание вероятности дефолта по кредиту является особо значимой задачей, так как финансовые организации заинтересованы в автоматизации решений, связанных с выдачей займа. Справиться с этим помогают методы машинного обучения, а именно, применяется машинное обучение для выбора характеристик заемщика, связанных с вероятностью невыплаты платежей этим заемщиком, и предсказание дефолта основываясь на отобранных характеристиках. Исходя из этих результатов, финансовая организация может подсчитать кредитный риск потенциального заемщика и решить, заинтересована ли она в выдаче кредита этому лицу.

В данной работе рассматривается датасет MoneyMe [0], содержащий более 200 характеристик более 12тыс. заемщиков, включая данные о произошедшем/не произошедшем дефолте.

Таким образом, целью работы является достижение способности научиться определять класс дефолта заемщика(дефолт/не дефолт) с помощью методов машинного обучения, основываясь на данных о заемщике и его кредитной истории.

Задачи:

1) основываясь на результатах предыдущих исследований по теме предсказания вероятности дефолта, выбрать и отрегулировать классификатор для обучения из открытой библиотеки Scikit_learn

2) справиться с проблемой несбалансированности целевых классов в датасете

3) применив метод кросс-валидации, обработать результаты классификатора, получить интересующие нас характеристики

4) задействовать эволюционный алгоритм выбора характеристик заемщика, влияющих на вероятность дефолта

5) сравнить с предыдущим результатом

Основная часть

Перед нами задача классификации дефолтных и не дефолтных займов.

1) Предобработка датасета

Характеристики(features) нашего датасета представлены как в виде двумерного массива чисел с плавающей точкой, в котором каждый столбец является непрерывным признаком (continuous feature),так и в виде категориальных признаков (categorical features), не имеющих числовых значений(напр. тип кредита). Категориальные данные требуют

предварительной обработки для того, чтобы модель могла с ними работать. Для этих целей обычно широко используется

Прямое кодирование (one-hot-encoding): идея, лежащая в основе прямого кодирования, заключается в том, чтобы заменить категориальную переменную одной или несколькими новыми признаками, которые могут принимать значения 0 и 1. Значения 0 и 1 придают смысл моделям в scikit-learn и с помощью дамми переменных мы можем выразить любое количество категорий, вводя по одному новому признаку для каждой категории.

Мы же используем Бинарное кодирование(Binaty encoding) из библиотеки category_encoders, похожее на OneHot, но хранящее каждую категорию как бинарную строку, таким образом уменьшая размерность датасета в сравнении с OneHotEncoding(что нам интересно в виду большого количества фич)

2) Масштабирование

Часто диапазон изменения переменной датасета сильно варьируется от одной переменной(feature) к другой, что приводит ухудшению работы алгоритма. Иными словами, многие алгоритмы, в том числе Random Forest, чувствительны к масштабированию данных. Чтобы преодолеть эту неприятность, воспользуемся нормализатором MinMaxScaler, который переводит значения переменных в <0, 1>.

3) Undersampling

В несбалансированных данных классификатор склонен игнорировать мьньший класс, так как это не приводит к сильной потере точности. Для улучшения способности модели предсказывать этот класс(в данном случае default), используют обработку тренировочных данных с помощью undersampling и oversampling методов. Возьмем RUS (random undersampler)

4) Выбор классификатора

Дерево решений(Decision Tree) - модель машинного обучения, по структуре представляющая собой «листья» и «ветки». На рёбрах («ветках») дерева решения записаны атрибуты, от которых зависит целевая функция, в «листьях» записаны значения целевой функции, а в остальных узлах — атрибуты, по которым различаются случаи. Чтобы классифицировать новый случай, надо спуститься по дереву до листа и выдать соответствующее значение.

Ансамбли (ensembles) – это методы, которые сочетают в себе множество моделей машинного обучения, чтобы в итоге получить более мощную модель.

Случайный лес(Random Forest) – это набор Decision Trees, где каждое дерево немного отличается от остальных. Идея случайного леса заключается в том, что каждое дерево может довольно хорошо прогнозировать, но скорее всего переобучается на части данных. Если мы построим много деревьев, которые хорошо работают и переобучаются с разной степенью, мы можем уменьшить переобучение путем усреднения их результатов. Уменьшение переобучения при сохранении прогнозной силы деревьев можно проиллюстрировать с помощью строгой математики.

параметры: n_estimators = количество деревьев в ансамбле, чем больше- тем выше точность max_features = количество отбираемых признаков для каждого узла, часто берется квадрат или логарифм всего количества фич n_jobs = распараллеливание random_state = устанавливаем для фиксации результата

атрибуты feature_importance - коэффициент важности каждого признака predict_proba = вероятность точки принадлежать к первому и второму классу, используется для регулирования порогового значения (threshold) вхождения эл-та в класс

Основываясь на результатах работы [1], согласно которой наиболее высокий показатель accuracy на тестовой выборке достигается с помощью ансамбля Random Forest Classifier, в данной работе используется этот классификатор.

5) Матрица ошибок(confusion matrix) -

$\begin{pmatrix} TN & FP \\ FN & TP \end{pmatrix}$, где

TN = правильно спрогнозироанный класс negative

TP = правильно спрогнозироанный класс positive

FP = неправильно спрогнозироанный класс negative

FN = неправильно спрогнозироанный класс positive

Матрица ошибок удобна для анализа полученного прогноза.

6) Перекрестная проверка (cross-validation) - статистический метод оценки обобщающей способности модели, который является более устойчивым и основательным, чем разбиение данных на обучающий и тестовый наборы. В перекрестной проверке данные разбиваются несколько раз и строится несколько моделей. Данный метод имеет ряд примуществ в сравнении с традиционным разделением данных на обучающий и тестовый наборы.

k-fold cross-validation: k – это задаваемое число блоков разбиения, как правило, 5 или 10. Данные сначала разбиваются на k частей (примерно) одинакового размера, называемых блоками (folds). Затем строится последовательность k моделей. Модель i обучается, используя блок i в качестве тестового набора, а остальные блоки выполняют роль обучающего набора, i $\in$ {1,2,..k}.

Однако мы используем

stratified k-fold cross-validation: разбиваем данные таким образом, чтобы пропорции классов в каждом блоке в точности соответствовали пропорциям классов в наборе данных, что позволяет получить более надежные оценки обобщающей способности.

7) Генетический алгоритм выбора признаков(Genetic algorithm): Генетический алгоритм моделирует процесс естественного отбора. Подмножеству признаков ставим в соответствие некоторый бинарный вектор (b1,b2,…,bn) где bi=1 значит, что i-й признак входит в признаковое подпространство и 0 значит что не входит (вектор является упрощенной математической моделью ДНК). Далее определяем операции кроссовера и мутации, генерируем случайный набор из K бинарных векторов. И генерируем "новое поколение": берем из набора случайные пары векторов и скрещиваем их при помощи кроссовера, после чего мутируем получившегося потомка. Из всех получившихся признаков (и старое и новое поколение) находим K оптимальных (дающих наилучший результат при обучении). После чего уже для них повторяем алгоритм до сходимости или по количеству итераций. K - параметр алгоритма.

In [0]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import train_test_split,  StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import MinMaxScaler
from imblearn.under_sampling import RandomUnderSampler
from category_encoders.binary import BinaryEncoder


# cross-validation with RF, th for threshold
def cv(X, y, th=0.5, undersampling=False):
    kf = StratifiedKFold(n_splits=5, random_state=0 )
    accuracy = 0
    matrices = []
    for train_index, test_index in kf.split(X, y):
        f2 = RandomForestClassifier(n_estimators=150, max_features=5, n_
jobs=-1,
                                    random_state=2)
        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]

        if undersampling:
            rus = RandomUnderSampler(random_state=0)
            X_train, y_train = rus.fit_resample(X_train, y_train)

        scaler = MinMaxScaler()
        X_train_transf = scaler.fit(X_train).transform(X_train)
        X_test_transf = scaler.transform(X_test)
        X_test_transf = pd.DataFrame(X_test_transf).fillna(0)
        X_train_transf = pd.DataFrame(X_train_transf).fillna(0)

        f2.fit(X_train_transf, y_train)
        score = f2.predict_proba(X_test_transf)[:, 1] > th
        accuracy += accuracy_score(y_test, score)
        conf = confusion_matrix(y_test, score)
        matrices.append(conf)
    s = matrices[0]
    for i in range(1, len(matrices)):
        s += matrices[i]
    print("summary confusion matrix:\n {}".format(s))
    print("total accuracy: {:.4f}".format(accuracy / 5))

    # график feature importance
def plot_feature_importance(model, X):
    n_features = X.shape[1]
    plt.barh(range(n_features), model.feature_importances_, align='cente
r')
    plt.yticks(np.arange(n_features), X.columns)
    plt.xlabel("feature importance")
    plt.ylabel("feature")

  # RF на признаках cols[]
def predict(X, y, cols ):
    new_data = X[cols]
    new_data = new_data.fillna(new_data.mean())

    X_train, X_test, y_train, y_test = train_test_split(new_data, y, ran
dom_state=0)

    scaler2 = MinMaxScaler()
    X_train_sc = scaler2.fit_transform(X_train)
    X_test_sc = scaler2.transform(X_test)
    X_test_sc = pd.DataFrame(X_test_sc).fillna(0)
```

```
    X_train_sc = pd.DataFrame(X_train_sc).fillna(0)

    rus2 = RandomUnderSampler(random_state=0)
    X_res, y_res = rus2.fit_resample(X_train_sc, y_train)

    forest2 = RandomForestClassifier(n_estimators=150, max_features=6, r
andom_state=0, n_jobs=-1)
    forest2.fit(X_res, y_res)
    preds2 = forest2.predict_proba(X_test_sc)[:, 1] > 0.7

    return forest2
```

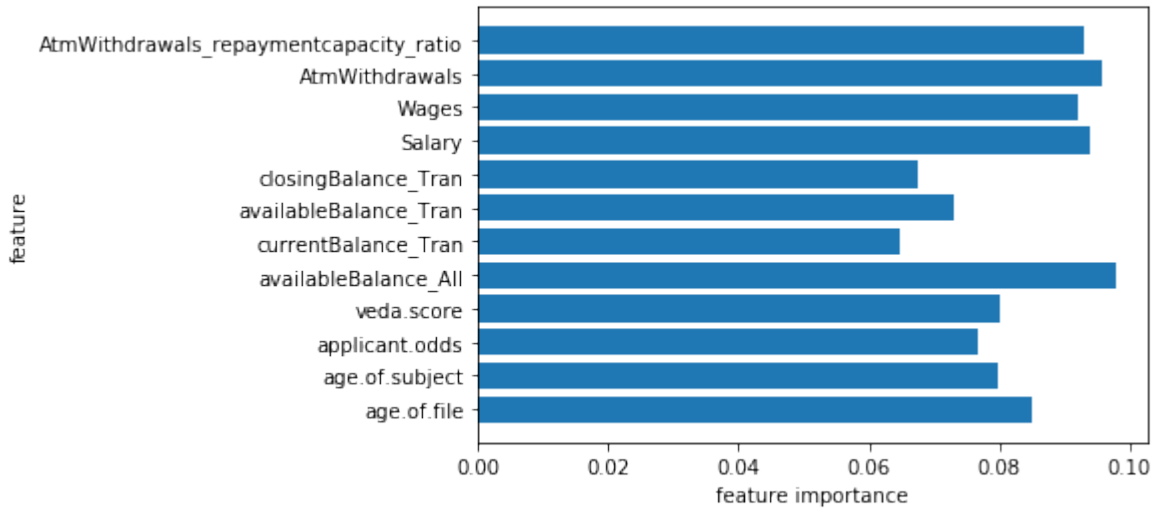
получаем feature importance: запускаем RF на всем датасете, после чего анализируем атрибут модели feature importance:

In [22]:
```
if __name__ == "__main__":
    file = '/content/drive/My Drive/applicationdata.csv'
    train_data = pd.read_csv(file)
    train_data = train_data.dropna(subset=['Isdefault']) # убираем пап из целе
вой переменной
    y = train_data.Isdefault                                    # целевая переменная
    train_data = train_data.drop(columns=['Isdefault','IsWrittenOff', 'M
axCdia',
                                    'ArrearsLevel','CustomerID', 'ATM_num']
)
    bin_train_data = train_data.fillna(method='backfill') # заполняем пап
    bin_train_data = bin_train_data.fillna(method='ffill')

    # handling categorical data in dataset
    encoder = BinaryEncoder()
    bin_train_data = encoder.fit_transform(bin_train_data)
    X = bin_train_data

# разделяем данные для обучения и теста
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_sta
te=0)

    # preprocessing data with scaling
    scaler = MinMaxScaler()
    X_train_transf = scaler.fit(X_train).transform(X_train)
    X_test_transf = scaler.transform(X_test)
    X_train_transf = pd.DataFrame(X_train_transf).fillna(0)
    X_test_transf = pd.DataFrame(X_test_transf).fillna(0)

    # undersampling
    rus = RandomUnderSampler(random_state=0)
    X_resampled, y_resampled = rus.fit_resample(X_train_transf, y_train)

# RF
    forest = RandomForestClassifier(n_estimators=100, random_state=0, n_
jobs=-1, max_features=9)
    forest.fit(X_resampled, y_resampled)

    # selecting the most important features
    cols = []
    for name, importance in zip(bin_train_data.columns, forest.feature_i
```

```
mportances_):
        if importance >= 0.008:
            cols.append(name)

    f = predict(train_data, y, cols)
    plot_feature_importance(f, train_data[cols])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/data.py:355
: RuntimeWarning: All-NaN slice encountered
  data_min = np.nanmin(X, axis=0)
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/data.py:356
: RuntimeWarning: All-NaN slice encountered
  data_max = np.nanmax(X, axis=0)
```

```
prediction by RF feature selection:
```



Интересующие нас переменные видны на графике. После регулировки порогового значения так, чтобы TP было весомым, получаем результат:

$\begin{pmatrix} 10314 & 377 \\ 2083 & 206 \end{pmatrix}$, total accuracy: 0.8105

In [23]:
```
cv(bin_train_data[cols], y, th=0.76, undersampling=True) # пороговое значение устанавливалось экспериментальным путем
```

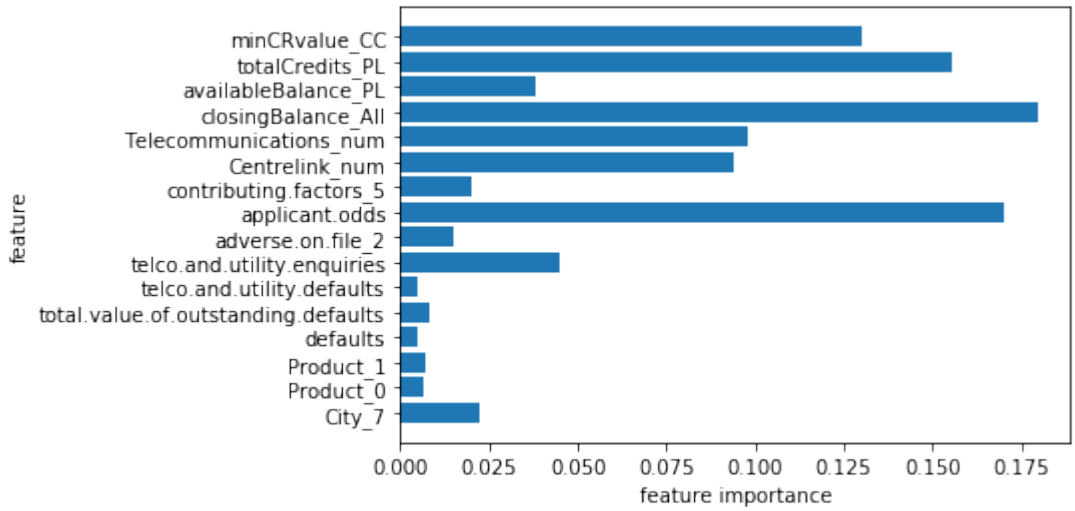
```
summary confusion matrix:
 [[10314   377]
 [ 2083   206]]
total accuracy: 0.8105
```

4.Для сравнения с результатами выбора переменных с помощью RF протестируем Genetic Algorithm for feature selection[1] - еще один мощный инструмент feature ingeneering. Для экономии места сразу возьмем отобранные им features в массив support и выведем результаты новой RF модели, действующей на данных с этими переменными (код GA взят с [4])

In [25]:
```
    forest3 = RandomForestClassifier(n_estimators=100, max_features=6, r
andom_state=0, n_jobs=-1)
    forest3.fit(X_resampled[:, support], y_resampled)
    most_important_columns = []
    columns = []
```

```
    for i in range(len(forest3.feature_importances_)):
        if support[i]:
            columns.append(bin_train_data.columns[i])
            if forest3.feature_importances_[i] >= 0.01:
              most_important_columns.append(bin_train_data.columns[i])
    forest2=predict(bin_train_data, y, most_important_columns)
    cv(bin_train_data[columns], y, th=0.66, undersampling=True)
    plot_feature_importance(forest2, bin_train_data[most_important_colum
ns])
```

```
summary confusion matrix:
 [[10114   577]
 [ 2080   209]]
total accuracy: 0.7953
```



видим, что результатом отбора является уже немного другое подмножество признаков, причем на нем мы получаем меньшую корректность при том же TP: $\begin{pmatrix} 10114 & 577 \\ 2080 & 209 \end{pmatrix}$, total accuracy: 0.7953

Таким образом, нам удалось получить данные о влиянии некоторых признаков на значение целевой переменной, а также достичь точности в 0.8105.

Ссылки и использованная литература

[0] dataset link : https://www.dropbox.com/sh/cwnp15hz6d7dk8n/AAAdtQx8-d-7j_qOH36qT-z1a?dl=0

[1] Genetic Algorithms as a Tool for Feature Selection in Machine Learning (Haleh Vafaie and Kenneth De Jong)

[2] Introduction to Machine Learning with Python (Andreas C. Müller & Sarah Guido)

[3] Two-stage consumer credit risk modelling using heterogeneous ensemble learning (Monika Papouskova, Petr Hajek)

[4] https://github.com/dawidkopczyk/genetic/blob/master/genetic.py