Лабораторна робота №7.
Алгоритм випадкового лісу в задачі кредитного скорінгу

Необхідно розв'язати задачу кредитного скорінга

Ознаки клієнта банку:

- Age - вік (дійсночислова)
- Income - місячний дохід (дійсночислова)
- BalanceToCreditLimit - відношення балансу на кредитній картці до ліміту за кредитом (дійсночислова)
- DIR - Debt-to-income Ratio (дійсночислова)
- NumLoans - кылькість позичок і кредитних ліній
- NumRealEstateLoans - кількість іпотек і позичок, пов'язаних з нерухомістю (натуральне число)
- NumDependents - кількість членів сім'ї, яких утримує клієнт, без врахування самого клієнту (натуральне число)
- Num30-59Delinquencies - кількість протермінувань виплат за кредитом від 30 до 59 днів (натуральне число)
- Num60-89Delinquencies - кількість протермінувань виплат за кредитом від 60 до 89 дній (натуральне число)
- Delinquent90 - чи були протермінування виплат за кредитом більше 90 днів (бінарний)

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
%matplotlib inline
```

Спочатку налаштуємо доступ до даних на google drive (якщо ви відкриваєте блокнот в google colab, а не на PC) шляхом монтування google drive

In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

Перевіримо шлях до папки з матеріалами лабораторної роботи на google drive. Якщо у вас шлях відрізняється то відредагуйте

In [ ]:
```
!ls gdrive/'My Drive'/TEACHING/IntroDataScience/intro_to_data_science/Lab_7_8/da
```

Перемістимо матеріали лабораторної роботи з google drive на віртуальну машину google colab

In [ ]:
```
!cp -a gdrive/'My Drive'/TEACHING/IntroDataScience/intro_to_data_science/Lab_7_8/
!ls
```

Завантажимо дані з використанням pandas

In [2]:
```
train_df = pd.read_csv('data/credit_scoring_train.csv', index_col='client_id')
test_df = pd.read_csv('data/credit_scoring_test.csv', index_col='client_id')
```

In [3]:
```
y = train_df['Delinquent90']
train_df.drop('Delinquent90', axis=1, inplace=True)
```

In [4]:
```
train_df.head()
```

Out[4]:

client_id	DIR	Age	NumLoans	NumRealEstateLoans	NumDependents	Num30-59Delinquencies	8
0	0.496289	49.1	13	0	0.0	2	
1	0.433567	48.0	9	2	2.0	1	
2	2206.731199	55.5	21	1	NaN	1	
3	886.132793	55.3	3	0	0.0	0	
4	0.000000	52.3	1	0	0.0	0	

Переглянемо кількість пропусків в кожній ознаці.

In [5]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 75000 entries, 0 to 74999
Data columns (total 9 columns):
DIR                     75000 non-null float64
Age                     75000 non-null float64
NumLoans                75000 non-null int64
NumRealEstateLoans      75000 non-null int64
NumDependents           73084 non-null float64
Num30-59Delinquencies   75000 non-null int64
Num60-89Delinquencies   75000 non-null int64
Income                  60153 non-null float64
BalanceToCreditLimit    75000 non-null float64
dtypes: float64(5), int64(4)
memory usage: 5.7 MB
```

In [6]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 75000 entries, 75000 to 149999
Data columns (total 9 columns):
DIR                     75000 non-null float64
Age                     75000 non-null float64
NumLoans                75000 non-null int64
NumRealEstateLoans      75000 non-null int64
NumDependents           72992 non-null float64
Num30-59Delinquencies   75000 non-null int64
Num60-89Delinquencies   75000 non-null int64
Income                  60116 non-null float64
BalanceToCreditLimit    75000 non-null float64
dtypes: float64(5), int64(4)
memory usage: 5.7 MB
```

Замінимо пропуски медіанними значеннями.

In [7]:
```
train_df['NumDependents'].fillna(train_df['NumDependents'].median(), inplace=Tru
train_df['Income'].fillna(train_df['Income'].median(), inplace=True)
test_df['NumDependents'].fillna(test_df['NumDependents'].median(), inplace=True)
test_df['Income'].fillna(test_df['Income'].median(), inplace=True)
```

Дерево рішень без налаштування параметрів

Обучите дерево решений максимальной глубины 3, используйте параметр random_state=17 для воспроизводимости результатов.

In [ ]:
```
first_tree = # Ваш код тут
first_tree.fit # Ваш код тут
```

Зробіть прогноз для тестової вибірки.

In [ ]:
```
first_tree_pred = first_tree # Ваш код тут
```

Запишемо прогноз у файл.

In [ ]:
```
def write_to_submission_file(predicted_labels, out_file,
                             target='Delinquent90', index_label="client_id"):
    # turn predictions into data frame and save as csv file
    predicted_df = pd.DataFrame(predicted_labels,
                                index = np.arange(75000,
                                                  predicted_labels.shape[0] + 75
                                columns=[target])
    predicted_df.to_csv(out_file, index_label=index_label)
```

In [ ]:
```
write_to_submission_file(first_tree_pred, 'credit_scoring_first_tree.csv')
```

Якщо прогнозувати ймовірності дефолту для клієнтів тестової вибірки, результат буде набагато кращим.

In [ ]:
```
first_tree_pred_probs = first_tree.predict_proba(test_df)[:, 1]
```

In [ ]:
```
write_to_submission_file # Ваш код тут
```

Дерево рішень без налаштування параметрів за допомогою GridSearch

Налаштуйте параметри дерева за допомогою `GridSearchCV` , подивіться на кращу комбінацію параметрів і середню якість на 5-кратній крос-валідації. Використовуйте параметр `random_state=17` (для відтворюваності результатів), не забувайте про розпаралелювання (`n_jobs=-1`).

In [ ]:
```
tree_params = {'max_depth': list(range(3, 8)),
               'min_samples_leaf': list(range(5, 13))}

locally_best_tree = GridSearchCV # Ваш код тут
locally_best_tree.fit # Ваш код тут
```

In [ ]:
```
locally_best_tree.best_params_, round(locally_best_tree.best_score_, 3)
```

Зробіть прогноз для тестової вибірки.

In [ ]:
```
tuned_tree_pred_probs = locally_best_tree # Ваш код тут
```

In [ ]:
```
write_to_submission_file # Ваш код тут
```

Випадковий ліс без настройки параметрів

Навчіть випадковий ліс з дерев необмеженої глибини, використовуйте параметр `random_state=17` для відтворюваності результатів.

In [ ]:
```
first_forest = # Ваш код тут
first_forest.fit # Ваш код тут
```

In [ ]:
```
first_forest_pred = first_forest # Ваш код тут
```

Зробіть прогноз для тестової вибірки.

In [ ]:
```
write_to_submission_file # Ваш код тут
```

Випадковий ліс з налаштуванням параметрів

Налаштуйте параметр `max_features` лісу за допомогою `GridSearhCV`, подивіться на кращу комбінацію параметрів і середню якість на 5-кратній крос-валідації. Використовуйте параметр random_state=17 (для відтворюваності результатів), не забувайте про розпаралелювання (n_jobs=-1).

In [ ]:
```
%%time
forest_params = {'max_features': np.linspace(.3, 1, 7)}

locally_best_forest = GridSearchCV # Ваш код тут
locally_best_forest.fit # Ваш код тут
```

In [ ]:
```
locally_best_forest.best_params_, round(locally_best_forest.best_score_, 3)
```

In [ ]:
```
tuned_forest_pred = locally_best_forest # Ваш код тут
```

In [ ]:
```
write_to_submission_file # Ваш код тут
```

Подивіться, як налаштований випадковий ліс оцінює важливість ознак за їх впливом на цільову ознаку. Подайте результати в наглядному вигляді за допомогою `DataFrame`.

In [ ]:
```
pd.DataFrame(locally_best_forest.best_estimator_.feature_importances_ # Ваш код
```

Як правило збільшення кількості дерев тільки покращує результат. Так що на останок навчіть випадковий ліс з 300 дерев зі знайденими кращими параметрами. Це може зайняти декілька хвилин.

In [ ]:
```
%%time
final_forest = RandomForestClassifier # Ваш код тут
final_forest.fit(train_df, y)
final_forest_pred = final_forest.predict_proba(test_df)[:, 1]
write_to_submission_file(final_forest_pred, 'credit_scoring_final_forest.csv')
```