rachittoshniwal / **ML-projects**

Code | Issues | Pull requests | Actions | Projects | Wiki | Security

main

ML-projects / credit risk / **loan approval.ipynb**

rachittoshniwal Add files via upload — History

1 contributor

2849 lines (2849 sloc) | 222 KB

Part 1

Some data wrangling

Some outlier removal based on domain knowledge

Use Column Transformer and Pipeline to streamline process

Use Randomized Search to find optimal set of parameters

Automate the procedure for multiple classifiers

Plot Precision-Recall Curve

Plot Learning Curve (for bias-variance tradeoff / check for overfitting-underfitting)

Part 2

Rectify existing model based on inferences from the learning curve and make a better one

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, learning_curve, R
andomizedSearchCV
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.metrics import plot_precision_recall_curve
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegres
sor
from sklearn.metrics import plot_confusion_matrix, confusion_matrix, cl
assification_report
from lightgbm import LGBMClassifier
```

In [2]:
```
df = pd.read_csv('credit_risk_dataset.csv')
df.head()
```

Out[2]:

	person_age	person_income	person_home_ownership	person_emp_length	lo
0	22	59000	RENT	123.0	PI
1	21	9600	OWN	5.0	EI
2	25	9600	MORTGAGE	1.0	M
3	23	65500	RENT	4.0	M
4	24	54400	RENT	8.0	M

In [3]:
```
dups = df.duplicated()
```

In [4]:
```
df[dups]
```

Out[4]:

	person_age	person_income	person_home_ownership	person_emp_lengt
15975	23	42000	RENT	5.0
15989	23	90000	MORTGAGE	7.0
15995	24	48000	MORTGAGE	4.0
16025	24	10000	RENT	8.0
16028	23	100000	MORTGAGE	7.0
...	...	...	...	...
32010	42	39996	MORTGAGE	2.0
32047	36	250000	RENT	2.0
32172	49	120000	MORTGAGE	12.0
32259	39	40000	OWN	4.0
32279	43	11340	RENT	4.0

165 rows × 12 columns

In [5]:
```
df.query("person_age==23 & person_income==42000 &\
person_home_ownership=='RENT' & loan_int_rate==9.99")
```

Out[5]:

	person_age	person_income	person_home_ownership	person_emp_lengt
6464	23	42000	RENT	5.0
15975	23	42000	RENT	5.0

In [6]:
```
df.shape
```

Out[6]: (32581, 12)

In [7]:
```
df.drop_duplicates(inplace=True)
```

In [8]:
```
df.shape
```

Out[8]: (32416, 12)

In [9]:
```
# X and y will be thought of as the entire training data
# X_test and y_test will be thought of as the out of sample data for mo
del evaluation

X, X_test, y, y_test = train_test_split(df.drop('loan_status', axis=1),
df['loan_status'],
```

```
                                        random_state=0,  test_size=0.2,
stratify=df['loan_status'],
                                        shuffle=True)
```

In [10]:
```
df['loan_status'].value_counts(normalize=True)
```

Out[10]:
```
0    0.781312
1    0.218688
Name: loan_status, dtype: float64
```

In [11]:
```
y.value_counts(normalize=True)
```

Out[11]:
```
0    0.781313
1    0.218687
Name: loan_status, dtype: float64
```

In [12]:
```
y_test.value_counts(normalize=True)
```

Out[12]:
```
0    0.781308
1    0.218692
Name: loan_status, dtype: float64
```

In [13]:
```
np.round(X.isna().sum()* 100 / X.shape[0], 3)
```

Out[13]:
```
person_age                    0.000
person_income                 0.000
person_home_ownership         0.000
person_emp_length             2.800
loan_intent                   0.000
loan_grade                    0.000
loan_amnt                     0.000
loan_int_rate                 9.614
loan_percent_income           0.000
cb_person_default_on_file     0.000
cb_person_cred_hist_length    0.000
dtype: float64
```

In [14]:
```
X.shape
```

Out[14]:
```
(25932, 11)
```

In [15]:
```
X.dropna().shape
```

Out[15]:
```
(22763, 11)
```

In [16]:
```
(25932-22763)/25932
```

Out[16]:
```
0.12220422643837729
```

In [17]:
```
X[['person_income', 'loan_amnt', 'loan_percent_income']].head()
```

Out[17]:

	person_income	loan_amnt	loan_percent_income
21415	48000	10000	0.21
12916	85000	7500	0.09
2938	125000	3000	0.02

19114	62000	2300	0.04
6057	48000	4200	0.09

In [18]:
```
X.drop('loan_percent_income', axis=1, inplace=True)
X_test.drop('loan_percent_income', axis=1, inplace=True)
```

In [19]:
```
for col in X:
    print(col, '--->', X[col].nunique())
    if X[col].nunique()<20:
        print(X[col].value_counts(normalize=True)*100)
    print()
```

```
person_age ---> 58

person_income ---> 3680

person_home_ownership ---> 4
RENT        50.320068
MORTGAGE    41.439149
OWN          7.916859
OTHER        0.323924
Name: person_home_ownership, dtype: float64

person_emp_length ---> 36

loan_intent ---> 6
EDUCATION            19.809502
MEDICAL              18.787598
VENTURE              17.542033
PERSONAL             16.878760
DEBTCONSOLIDATION    15.968687
HOMEIMPROVEMENT      11.013420
Name: loan_intent, dtype: float64

loan_grade ---> 7
A    32.932284
B    32.126330
C    19.902052
D    11.121394
E     3.004010
F     0.732685
G     0.181243
Name: loan_grade, dtype: float64

loan_amnt ---> 710

loan_int_rate ---> 346

cb_person_default_on_file ---> 2
N    82.392411
Y    17.607589
Name: cb_person_default_on_file, dtype: float64

cb_person_cred_hist_length ---> 29
```

In [20]:
```
X.describe()
```

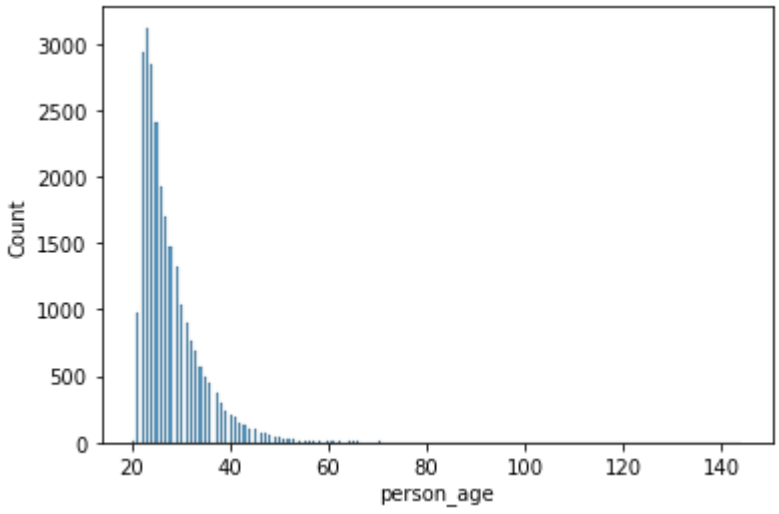
Out[20]:

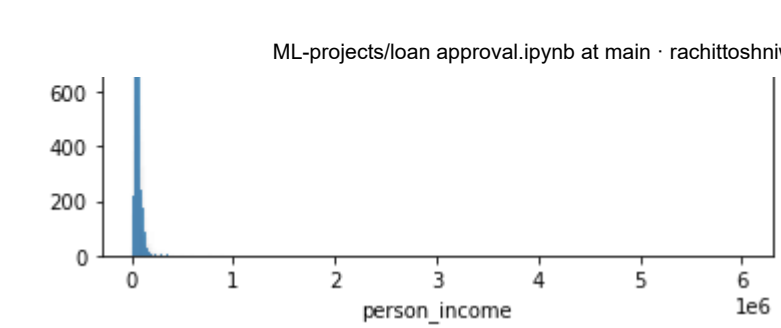
	person_age	person_income	person_emp_length	loan_amnt	loan_in
count	25932.000000	2.593200e+04	25206.000000	25932.000000	23439.0
mean	27.721155	6.589884e+04	4.811315	9611.395187	11.0137
std	6.382311	6.333831e+04	4.172822	6339.054572	3.24010
min	20.000000	4.000000e+03	0.000000	500.000000	5.42000
25%	23.000000	3.849500e+04	2.000000	5000.000000	7.90000
50%	26.000000	5.500000e+04	4.000000	8000.000000	10.9900
75%	30.000000	7.900000e+04	7.000000	12250.000000	13.4800
max	144.000000	6.000000e+06	123.000000	35000.000000	23.2200

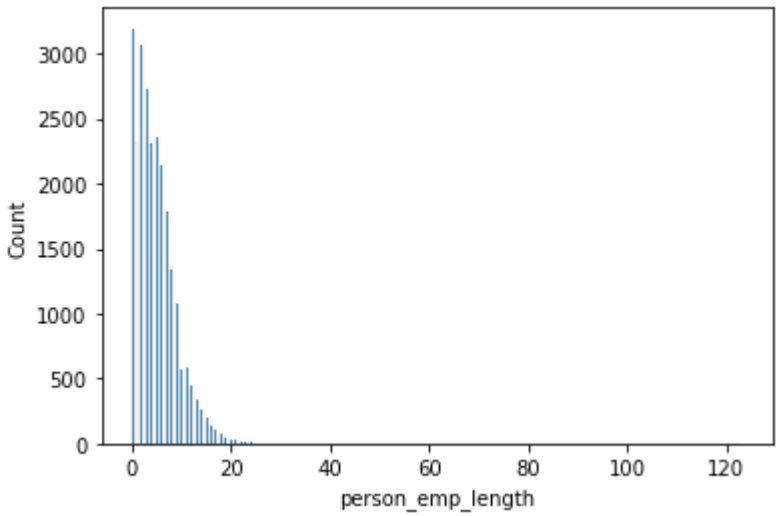
In [21]:
```
num_cols = [col for col in X if X[col].dtypes != 'O']
num_cols
```

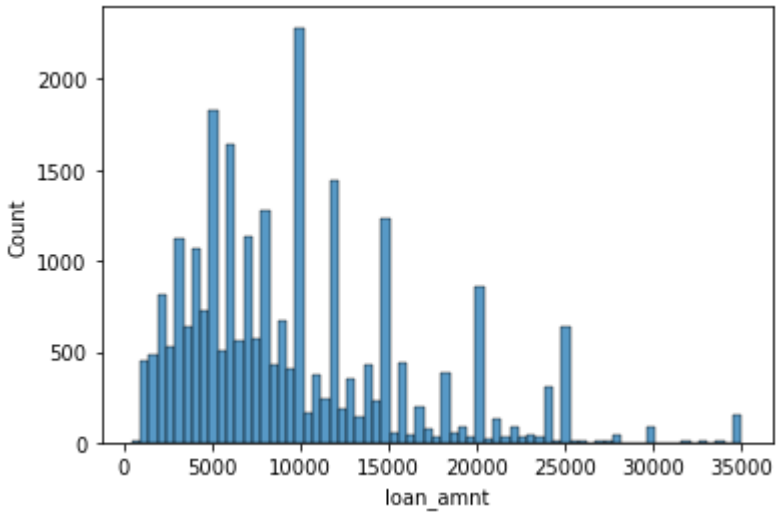
Out[21]:
```
['person_age',
 'person_income',
 'person_emp_length',
 'loan_amnt',
 'loan_int_rate',
 'cb_person_cred_hist_length']
```

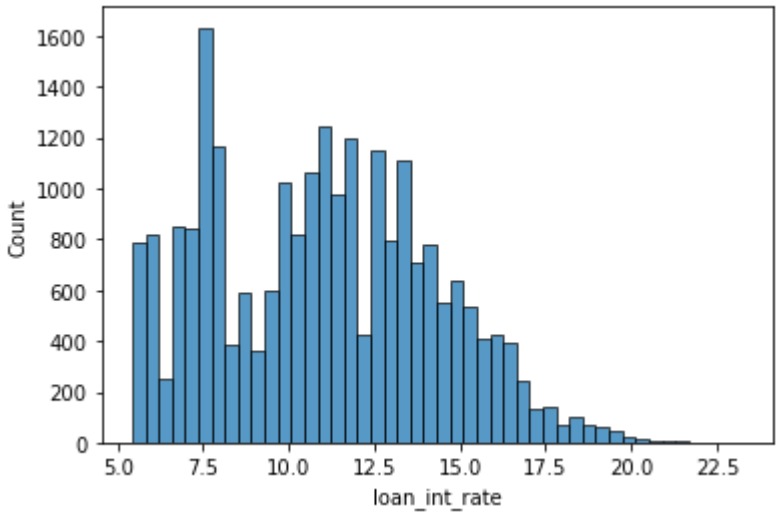
In [22]:
```
for col in num_cols:
    sns.histplot(X[col])
    plt.show()
```

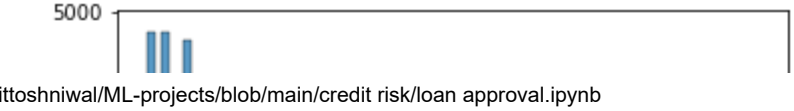


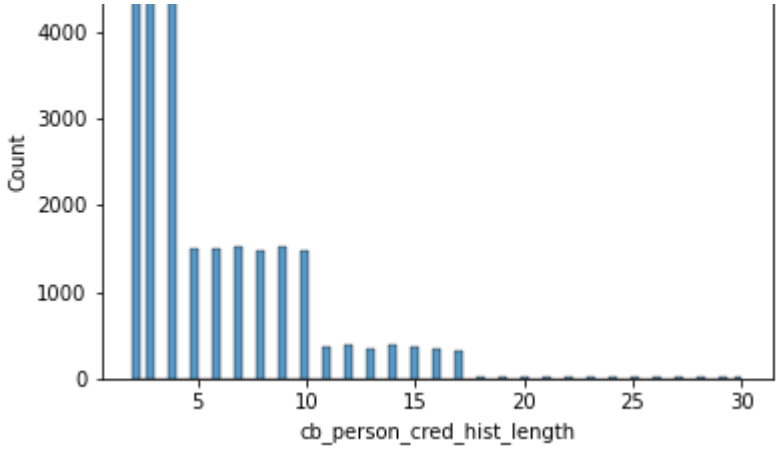

600
400
200
0
0
1
2
3
4
5
6
1e6
person_income


3000
2500
2000
1500
1000
500
0
Count
0
20
40
60
80
100
120
person_emp_length


2000
1500
1000
500
0
Count
0
5000
10000
15000
20000
25000
30000
35000
loan_amnt


1600
1400
1200
1000
800
600
400
200
0
Count
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
loan_int_rate


5000



In [23]:
```
X.loc[X['person_age']>=80, :]
```

Out[23]:

	person_age	person_income	person_home_ownership	person_emp_lengt
32422	80	64000	RENT	7.0
81	144	250000	RENT	4.0
32416	94	24000	RENT	1.0
747	123	78000	RENT	7.0
183	144	200000	MORTGAGE	4.0
575	123	80004	RENT	2.0
32506	84	94800	MORTGAGE	2.0
32297	144	6000000	MORTGAGE	12.0

In [24]:
```
X = X.loc[X['person_age']<80, :]
```

In [25]:
```
X.shape
```

Out[25]: (25924, 10)

In [26]:
```
X.loc[X['person_emp_length']>=66, :]
```

Out[26]:

	person_age	person_income	person_home_ownership	person_emp_length
210	21	192000	MORTGAGE	123.0
0	22	59000	RENT	123.0

In [27]:
```
df.query("person_age<=person_emp_length+14")
```

Out[27]:

	person_age	person_income	person_home_ownership	person_emp_length
0	22	59000	RENT	123.0
210	21	192000	MORTGAGE	123.0

In [28]:
```
X = X.loc[(X['person_emp_length']<66) | (X['person_emp_length'].isna
()), :]
```

In [29]:
```
# since we've removed some data from X, we need to pass on these updati
ons to y as well,
# as y doesn't know some of its corresponding X's have been deleted.
y = y[X.index]
```

In [30]:
```
cat_cols = [col for col in X if X[col].dtypes == 'O']
cat_cols
```

Out[30]:
```
['person_home_ownership',
 'loan_intent',
 'loan_grade',
 'cb_person_default_on_file']
```

In [31]:
```
num_pipe = Pipeline([
    ('impute', IterativeImputer()),
    ('scale', StandardScaler()),
])
```

In [32]:
```
ct = ColumnTransformer([
    ('num_pipe', num_pipe, num_cols),
    ('cat_cols', OneHotEncoder(sparse=False, handle_unknown='ignore'),
cat_cols)
], remainder='passthrough')
```

In [ ]:
```

```

In [33]:
```
grid = {
    RandomForestClassifier(random_state=0, n_jobs=-1, class_weight='bal
anced'):
    {'model__n_estimators':[300,400,500],
     'coltf__num_pipe__impute__estimator': [LinearRegression(), RandomF
orestRegressor(random_state=0),
                                           KNeighborsRegressor()]},

    LGBMClassifier(class_weight='balanced', random_state=0, n_jobs=-1):
    {'model__n_estimators':[300,400,500],
     'model__learning_rate':[0.001,0.01,0.1,1,10],
     'model__boosting_type': ['gbdt', 'goss', 'dart'],
     'coltf__num_pipe__impute__estimator':[LinearRegression(), RandomFo
restRegressor(random_state=0),
                                           KNeighborsRegressor()]},
}
```

In [34]:
```
for clf, param in grid.items():
    print(clf)
    print('-'*50)
    print(param)
    print('\n')
```

```
RandomForestClassifier(class_weight='balanced', n_jobs=-1, random_state
=0)
```

```
--------------------------------------------------
{'model__n_estimators': [300, 400, 500], 'coltf__num_pipe__impute__esti
mator': [LinearRegression(), RandomForestRegressor(random_state=0), KNe
ighborsRegressor()]}


LGBMClassifier(class_weight='balanced', random_state=0)
--------------------------------------------------
{'model__n_estimators': [300, 400, 500], 'model__learning_rate': [0.00
1, 0.01, 0.1, 1, 10], 'model__boosting_type': ['gbdt', 'goss', 'dart'],
'coltf__num_pipe__impute__estimator': [LinearRegression(), RandomForest
Regressor(random_state=0), KNeighborsRegressor()]}
```

In [35]:
```
full_df = pd.DataFrame()
best_algos = {}

for clf, param in grid.items():
    pipe = Pipeline([
    ('coltf', ct),
    ('model', clf)
])

    gs = RandomizedSearchCV(estimator=pipe, param_distributions=param,
scoring='accuracy',
                            n_jobs=-1, verbose=3, n_iter=4, random_stat
e=0)

    gs.fit(X, y)

    all_res = pd.DataFrame(gs.cv_results_)

    temp = all_res.loc[:, ['params', 'mean_test_score']]
    algo_name = str(clf).split('(')[0]
    temp['algo'] = algo_name

    full_df = pd.concat([full_df, temp], ignore_index=True)
    best_algos[algo_name] = gs.best_estimator_
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

In [36]:
```
full_df.sort_values('mean_test_score', ascending=False)
```

Out[36]:

	params	mean_test_score	algo
3	{'model__n_estimators': 400, 'coltf__num_pipe_...	0.922151	RandomForestClassifier
0	{'model__n_estimators': 400, 'coltf__num_pipe_...	0.921727	RandomForestClassifier
1	{'model__n_estimators': 500, 'coltf__num_pipe_...	0.921688	RandomForestClassifier
2	{'model__n_estimators': 400, 'coltf__num_pipe_...	0.921650	RandomForestClassifier

7	{'model__n_estimators': 300, 'model__learning_...	0.908804	LGBMClassifier
4	{'model__n_estimators': 300, 'model__learning_...	0.869339	LGBMClassifier
5	{'model__n_estimators': 300, 'model__learning_...	0.868683	LGBMClassifier
6	{'model__n_estimators': 300, 'model__learning_...	0.863784	LGBMClassifier

In [37]:
```
full_df.sort_values('mean_test_score', ascending=False).iloc[0, 0]
```

Out[37]:
```
{'model__n_estimators': 400,
 'coltf__num_pipe__impute__estimator': RandomForestRegressor(random_sta
te=0)}
```

In [38]:
```
be = best_algos['RandomForestClassifier']
be
```

Out[38]:
```
Pipeline(steps=[('coltf',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('num_pipe',
                                                  Pipeline(steps=[('imp
ute',
                                                                   Iter
ativeImputer(estimator=RandomForestRegressor(random_state=0))),
                                                                  ('sca
le',
                                                                   Stan
dardScaler())]),
                                                  ['person_age',
                                                   'person_income',
                                                   'person_emp_length',
                                                   'loan_amnt', 'loan_i
nt_rate',
                                                   'cb_person_cred_hist
_length']),
                                                 ('cat_cols',
                                                  OneHotEncoder(handle_
unknown='ignore',
                                                                sparse=
False),
                                                  ['person_home_ownersh
ip',
                                                   'loan_intent', 'loan
_grade',
                                                   'cb_person_default_o
n_file'])])),
                ('model',
                 RandomForestClassifier(class_weight='balanced',
                                        n_estimators=400, n_jobs=-1,
                                        random_state=0))])
```

In [39]:
```
be.fit(X, y)
```

Out[39]:
```
Pipeline(steps=[('coltf',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('num_pipe',
                                                  Pipeline(steps=[('imp
ute',
                                                                   Iter
ativeImputer(estimator=RandomForestRegressor(random_state=0))),
                                                                   ('sca
le',
                                                                   Stan
dardScaler())]),
                                                  ['person_age',
                                                   'person_income',
                                                   'person_emp_length',
                                                   'loan_amnt', 'loan_i
nt_rate',
                                                   'cb_person_cred_hist
_length']),
                                                 ('cat_cols',
                                                  OneHotEncoder(handle_
unknown='ignore',
                                                                sparse=
False),
                                                  ['person_home_ownersh
ip',
                                                   'loan_intent', 'loan
_grade',
                                                   'cb_person_default_o
n_file'])])),
                ('model',
                 RandomForestClassifier(class_weight='balanced',
                                        n_estimators=400, n_jobs=-1,
                                        random_state=0))])
```

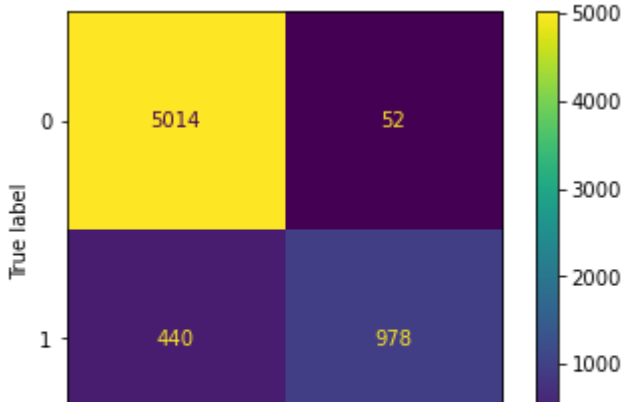
In [40]:
```
preds = be.predict(X_test)
```

In [41]:
```
confusion_matrix(y_test, preds)
```

Out[41]:
```
array([[5014,   52],
       [ 440,  978]], dtype=int64)
```

In [42]:
```
plot_confusion_matrix(be, X_test, y_test)
```

Out[42]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x258
d7485f40>
```



In [43]:
```
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95      5066
           1       0.95      0.69      0.80      1418

    accuracy                           0.92      6484
   macro avg       0.93      0.84      0.88      6484
weighted avg       0.93      0.92      0.92      6484
```

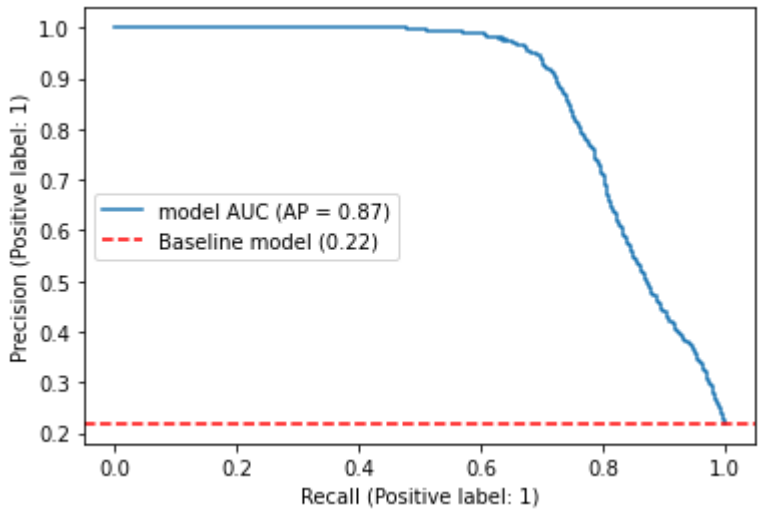
In [44]:
```
be.score(X_test, y_test)
```

Out[44]: 0.9241209130166563

precision recall curve

In [45]:
```
plot_precision_recall_curve(estimator=be, X=X_test, y=y_test, name='mod
el AUC')
baseline = y_test.sum() / len(y_test)
plt.axhline(baseline, ls='--', color='r', label=f'Baseline model ({roun
d(baseline,2)})')
plt.legend(loc='best')
```

Out[45]: <matplotlib.legend.Legend at 0x258dbecc8b0>



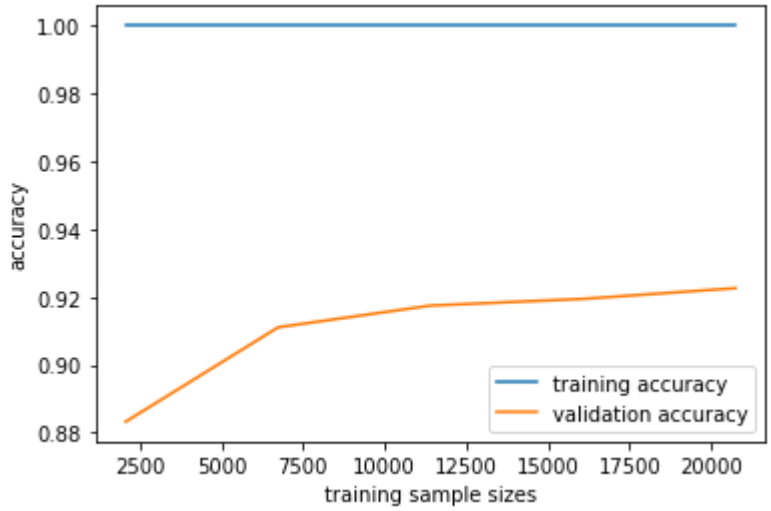
learning curve

In [46]:
```
a, b, c = learning_curve(be, X, y, n_jobs=-1, scoring='accuracy')
```

In [47]:
```
plt.plot(a, b.mean(axis=1), label='training accuracy')
plt.plot(a, c.mean(axis=1),  label='validation accuracy')
plt.xlabel('training sample sizes')
```

```
plt.ylabel('accuracy')
plt.legend()
```

Out[47]: <matplotlib.legend.Legend at 0x258d8c27bb0>



Overfitting:

1. High training accuracy (--- low bias)
2. Low testing/ validation accuracy (--- high variance)
3. Big gap between training and validation curves (--- high variance)
4. Overfitting makes a very complex model and learns even the "noise" in the data, which is undesirable

In [ ]:

Part 2

In [ ]:

Remedial measures:

1. Add more training samples, if possible, to allow the model to learn better

1. Working with data at hand:

   Make a simpler model / reduce complexity of model:

- try reducing number of features
- try increasing regularization (lambda)
- try pruning the decision trees

In [ ]:

In [48]:
```
grid = {

    RandomForestClassifier(random_state=0, n_jobs=-1, class_weight='bal
```

```
    RandomForestClassifier(random_state=0, n_jobs=-1, class_weight='bal
anced'):
    {'model__n_estimators':[100,200,300],
     'model__max_depth':[5, 9, 13],
     'model__min_samples_split':[4,6,8],
     'coltf__num_pipe__impute__estimator': [LinearRegression(), RandomF
orestRegressor(random_state=0),
                                            KNeighborsRegressor()]},

#     LGBMClassifier(class_weight='balanced', random_state=0, n_jobs=-
1):
#     {'model__n_estimators':[100,200,300],
#      'model__max_depth':[5, 9, 13],
#      'model__num_leaves': [7,15,31],
#      'model__learning_rate':[0.0001,0.001,0.01,0.1,],
#      'model__boosting_type': ['gbdt', 'goss', 'dart'],
#      'coltf__num_pipe__impute__estimator':[LinearRegression(), Random
ForestRegressor(random_state=0),
#                                            KNeighborsRegressor()]}
}
```

In [49]:
```
for clf, param in grid.items():
    print(clf)
    print('-'*50)
    print(param)
    print('\n')
```

```
RandomForestClassifier(class_weight='balanced', n_jobs=-1, random_state
=0)
--------------------------------------------------
{'model__n_estimators': [100, 200, 300], 'model__max_depth': [5, 9, 1
3], 'model__min_samples_split': [4, 6, 8], 'coltf__num_pipe__impute__es
timator': [LinearRegression(), RandomForestRegressor(random_state=0), K
NeighborsRegressor()]}
```

In [50]:
```
full_df = pd.DataFrame()
best_algos = {}

for clf, param in grid.items():
    pipe = Pipeline([
    ('coltf', ct),
    ('model', clf)
])

    gs = RandomizedSearchCV(estimator=pipe, param_distributions=param,
scoring='accuracy',
                            n_jobs=-1, verbose=3, n_iter=4)

    gs.fit(X, y)

    all_res = pd.DataFrame(gs.cv_results_)

    temp = all_res.loc[:, ['params', 'mean_test_score']]
    algo_name = str(clf).split('(')[0]
    temp['algo'] = algo_name
```

```
    full_df = pd.concat([full_df, temp])
    best_algos[algo_name] = gs.best_estimator_
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

In [51]:
```
full_df.sort_values('mean_test_score', ascending=False)
```

Out[51]:

	params	mean_test_score	algo
0	{'model__n_estimators': 100, 'model__min_sampl...	0.911620	RandomForestClassifier
1	{'model__n_estimators': 100, 'model__min_sampl...	0.908186	RandomForestClassifier
3	{'model__n_estimators': 200, 'model__min_sampl...	0.907878	RandomForestClassifier
2	{'model__n_estimators': 300, 'model__min_sampl...	0.907492	RandomForestClassifier

In [52]:
```
be = best_algos['RandomForestClassifier']
be
```

Out[52]:
```
Pipeline(steps=[('coltf',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('num_pipe',
                                                  Pipeline(steps=[('imp
ute',
                                                                   Iter
ativeImputer(estimator=LinearRegression())),
                                                                  ('sca
le',
                                                                   Stan
dardScaler())]),
                                                  ['person_age',
                                                   'person_income',
                                                   'person_emp_length',
                                                   'loan_amnt', 'loan_i
nt_rate',
                                                   'cb_person_cred_hist
_length']),
                                                 ('cat_cols',
                                                  OneHotEncoder(handle_
unknown='ignore',
                                                                sparse=
False),
                                                  ['person_home_ownersh
ip',
                                                   'loan_intent', 'loan
_grade',
                                                   'cb_person_default_o
n_file'])])),
                ('model',
                 RandomForestClassifier(class_weight='balanced', max_de
pth=13,
                                        min_samples_split=4, n_jobs=-1,
```

```
                                                  random_state=0))])
```

In [53]:
```
be.fit(X, y)
```

Out[53]:
```
Pipeline(steps=[('coltf',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('num_pipe',
                                                  Pipeline(steps=[('imp
ute',
                                                                   Iter
ativeImputer(estimator=LinearRegression())),
                                                                  ('sca
le',
                                                                   Stan
dardScaler())]),
                                                  ['person_age',
                                                   'person_income',
                                                   'person_emp_length',
                                                   'loan_amnt', 'loan_i
nt_rate',
                                                   'cb_person_cred_hist
_length']),
                                                 ('cat_cols',
                                                  OneHotEncoder(handle_
unknown='ignore',
                                                                sparse=
False),
                                                  ['person_home_ownersh
ip',
                                                   'loan_intent', 'loan
_grade',
                                                   'cb_person_default_o
n_file'])])),
                ('model',
                 RandomForestClassifier(class_weight='balanced', max_de
pth=13,
                                        min_samples_split=4, n_jobs=-1,
                                        random_state=0))])
```

In [54]:
```
preds = be.predict(X_test)
```