In [1]:
```
%matplotlib inline
import matplotlib
import seaborn as sns
matplotlib.rcParams['savefig.dpi'] = 144
```

In [2]:
```
from static_grader import grader
```

ML Miniproject

Introduction

The objective of this miniproject is to exercise your ability to create effective machine learning models for making predictions. We will be working with credit card data from Taiwan, predicting whether customers will default based on their recent billing data as well as demographics.

Scoring

In this miniproject you will submit the predictions of your model to the grader. The grader will assess the performance of your model using a scoring metric, comparing it against the score of a reference model. We will use the [average precision score (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html)](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html). If your model performs better than the reference solution, then you can score higher than 1.0.

Downloading the data

We can download the data set from Amazon S3:

In [3]:
```
!mkdir data
!aws s3 sync s3://dataincubator-wqu/mldata/ ./data
```

```
mkdir: cannot create directory 'data': File exists
```

We'll load the data into a Pandas DataFrame and pop out the target labels.

In [4]:
```
import numpy as np
import pandas as pd
```

In [5]:
```
data = pd.read_csv('./data/UCI_Credit_Card_train.csv', index_col=False)
target = data.pop('default.payment.next.month')

test = pd.read_csv('./data/UCI_Credit_Card_test.csv', index_col=False)
```

In [6]:
```
data.head()
```

Out[6]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BIL
0	50000.0	2	2	1	34	0	0	2	0	0	...	
1	80000.0	1	2	1	43	0	0	0	0	0	...	
2	200000.0	1	1	1	36	0	0	2	2	2	...	
3	280000.0	2	2	2	50	-1	-1	-1	-1	-1	...	
4	150000.0	2	2	1	51	0	0	0	0	0	...	

5 rows × 23 columns

In [7]:
```
target.head()
```

Out[7]:
```
0    0
1    0
2    1
3    0
4    0
Name: default.payment.next.month, dtype: int64
```

In [8]:
```
test.head()
```

Out[8]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BIL
0	10000.0	2	2	1	44	0	0	0	-2	-2	...	
1	90000.0	2	1	2	33	0	0	0	0	0	...	
2	140000.0	1	3	2	29	0	0	0	0	0	...	
3	340000.0	2	1	2	37	-1	0	-1	0	0	...	
4	60000.0	2	2	2	41	0	0	0	0	0	...	

5 rows × 23 columns

Question 1: billing_model

The most predictive aspect of the data set is the customer's billing history. Build a simple model that predicts whether a customer will default based only on the billing data. The model should implement a fit method that receives a DataFrame with the fields `'LIMIT_BAL'`, `'PAY_x'` (x = 0--6, except for 1), `'BILL_AMTx'` (x = 1--6), and `'PAY_AMTx'` (x = 1--6) as its feature matrix and the target labels. The model should also implement a predict method that receives the same features and returns *predicted label probabilities*. In most `sklearn` estimators this will be called `predict_proba`. It is important that you return predicted probabilities for compatibility with the ROC AUC metric.

In [11]:
```
from sklearn.preprocessing import StandardScaler

scaler1=StandardScaler()
data=pd.DataFrame(scaler1.fit_transform(data), columns=data.columns)

scaler2=StandardScaler()
test=pd.DataFrame(scaler2.fit_transform(test), columns=test.columns)
```

In [15]:
```
df=data.drop(['SEX', 'EDUCATION', 'MARRIAGE','AGE'], axis=1)
```

In [13]:
```
from sklearn.linear_model import LogisticRegression

billing_model=LogisticRegression()
billing_model.fit(df, target)

test_df=test.drop(['SEX', 'EDUCATION', 'MARRIAGE','AGE'], axis=1)
predictions=billing_model.predict_proba(test_df)
```

In [14]:
```
def bill_predictions():
    return predictions[:, 1]
```

In [ ]:
```

```

In [ ]:
```
billing_data = ...
```

In [ ]:
```
def bill_predictions():
    return np.random.random(3000)
```

In [15]:
```
grader.score('ml__billing_model', bill_predictions)
```

```
==================
Your score:  0.898219113148
==================
```

Question 2: balanced_billing

Default is rare, but we want to be sure to catch likely defaults before they happen; that is, we want high recall. What is the recall of your model? It may suffer due to class imbalance. Investigate the recall of your model and try to optimize it by creating a strategy to deal with class imbalance in the data set.

When you've updated your model, submit its `predict_proba` method to the grader.

In [16]:
```
from sklearn.model_selection import train_test_split

from sklearn.utils import shuffle

scaler=StandardScaler()

df_trans=scaler.fit_transform(df)

df=pd.DataFrame(df_trans, columns=df.columns)
```

In [17]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(*(df, target), test_size=0.3)
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import GridSearchCV

model = LogisticRegression()

gs = GridSearchCV(model,
                  {'penalty': ['l1', 'l2'],
                  'C': [.001, .01, .1]},
                      cv=5,
                      n_jobs=2,
                      scoring='neg_mean_squared_error')

gs.fit(X_train, y_train)

print gs.best_params_

gs.best_estimator_

model=gs.best_estimator_
model.fit(X_train, y_train)

predictions=model.predict(X_test)

from sklearn import metrics

metrics.recall_score(predictions, y_test)
```

```
{'penalty': 'l2', 'C': 0.1}
```

Out[17]:
```
0.6998313659359191
```

In [18]:
```
def bal_bill_predictions():
    return model.predict_proba(test_df)[:, 1]
```

In [ ]:

In [ ]:

In [ ]:
```
def bal_bill_predictions():
    return np.random.random(3000)
```

In [19]:
```

grader.score('ml__balanced_billing_model', bal_bill_predictions)
```

```
==================
Your score:  0.902307318638
==================
```

Question 3: demo_model

Billing data would not be available for prospective customers, but we may want to predict their risk of default if given a line of credit. Construct a model that only takes into account the fields `'SEX'`, `'EDUCATION'`, `'MARRIAGE'`, `'AGE'`, and `'LIMIT_BAL'` (which the creditor controls/knows in advance) to predict default.

In [24]:
```
df= data[['LIMIT_BAL', 'AGE']]
df.head()
```

Out[24]:

	LIMIT_BAL	AGE
0	50000.0	34
1	80000.0	43
2	200000.0	36
3	280000.0	50
4	150000.0	51

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

Out[25]:
```
(27000, 2)
```

In [18]:
```
data['SEX'].unique()
```

Out[18]:
```
array([2, 1])
```

In [19]:
```
data['EDUCATION'].unique()
```

Out[19]:
```
array([2, 1, 3, 5, 4, 6, 0])
```

In [20]:
```
data['MARRIAGE'].unique()
```

Out[20]:
```
array([1, 2, 3, 0])
```

In [28]:
```
sex = pd.get_dummies(data['SEX'], prefix = 'SEX')
sex.head()
```

Out[28]:

	SEX_1	SEX_2
0	0	1
1	1	0
2	1	0
3	0	1
4	0	1

In [26]:
```
sex.shape
```

Out[26]: (27000, 2)

In [29]:
```
ed = pd.get_dummies(data['EDUCATION'], prefix = 'EDUCATION')
ed.head()
```

Out[29]:

	EDUCATION_0	EDUCATION_1	EDUCATION_2	EDUCATION_3	EDUCATION_4	EDUCATION_5	EDI
0	0	0	1	0	0	0	
1	0	0	1	0	0	0	
2	0	1	0	0	0	0	
3	0	0	1	0	0	0	
4	0	0	1	0	0	0	

In [30]:
```
mar = pd.get_dummies(data['MARRIAGE'], prefix = 'MARRIAGE' )
mar.head()
```

Out[30]:

	MARRIAGE_0	MARRIAGE_1	MARRIAGE_2	MARRIAGE_3
0	0	1	0	0
1	0	1	0	0
2	0	1	0	0
3	0	0	1	0
4	0	1	0	0

In [31]:
```
df_2 = pd.concat([df, sex, ed, mar], axis=1)
df_2.head()
```

Out[31]:

	LIMIT_BAL	AGE	SEX_1	SEX_2	EDUCATION_0	EDUCATION_1	EDUCATION_2	EDUCATION_3
0	50000.0	34	0	1	0	0	1	0
1	80000.0	43	1	0	0	0	1	0
2	200000.0	36	1	0	0	1	0	0
3	280000.0	50	0	1	0	0	1	0
4	150000.0	51	0	1	0	0	1	0

In [36]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
df_2_trans=scaler.fit_transform(df_2)

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
model = LogisticRegression()

gs = GridSearchCV(model,
                  {'penalty': ['l1', 'l2'],
                  'C': [.00000001, .000001, .00001, .0001, .001, .01]},
                  cv=5,
                  n_jobs=2,
                  scoring='neg_mean_squared_error')
gs.fit(df_2_trans, target)
print gs.best_params_

model=LogisticRegression(penalty='l1', C=.000001)
model.fit(df_2_trans, target)
```

```
{'penalty': 'l1', 'C': 1e-08}
```

Out[36]:
```
LogisticRegression(C=1e-06, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l1', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [33]:
```
test_df = test[['LIMIT_BAL', 'AGE']]
```

In [34]:
```
test_sex = pd.get_dummies(test['SEX'], prefix = 'SEX')
test_ed = pd.get_dummies(test['EDUCATION'], prefix = 'EDUCATION')
test_mar = pd.get_dummies(test['MARRIAGE'], prefix = 'MARRIAGE')
```

In [35]:
```
test_df_2 = pd.concat([test_df, test_sex, test_ed, test_mar], axis=1)
test_df_2.head()
```

Out[35]:

ATION_0	EDUCATION_1	EDUCATION_2	EDUCATION_3	EDUCATION_4	EDUCATION_5	EDUCATION_6
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0

In [37]:
```
model.predict_proba(test_df_2)
```

Out[37]:
```
array([[ 0.5,  0.5],
       [ 0.5,  0.5],
       [ 0.5,  0.5],
       ...,
       [ 0.5,  0.5],
       [ 0.5,  0.5],
       [ 0.5,  0.5]])
```

In [38]:
```
def demo_predictions():
    return model.predict_proba(test_df_2)[:, 1]
```

In [ ]:
```

```

In [ ]:
```
def demo_predictions():
    return np.random.random(3000)
```

In [39]:
```
grader.score('ml__demo_model', demo_predictions)
```

```
==================
Your score:  2.12992424831
==================
```

Question 4: ensemble_model

Let's combine the output of our two models in a simple ensemble. That is, take the predicted probabilities of your model based on billing data and your model based on demographic data as inputs for a final estimator that combines them (maybe a simple logistic regression, for instance).

You will need to use pipelines and feature unions to accomplish this, because the grader will expect a model that accepts the full feature matrix as input.

In [16]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(*(df, target), test_size=0.3)

pipeline = Pipeline([
    ('sgd_lr_1', SGDClassifier(loss='log')),('sgd_lr', SGDClassifier(loss='log'))
])
```

In [18]:
```
pipeline.fit(X_train, y_train)
```

```
/opt/conda/lib/python2.7/site-packages/sklearn/utils/deprecation.py:70: Depreca
tionWarning: Function transform is deprecated; Support to use estimators as fea
ture selectors will be removed in version 0.19. Use SelectFromModel instead.
  warnings.warn(msg, category=DeprecationWarning)
```

Out[18]:
```
Pipeline(steps=[('sgd_lr_1', SGDClassifier(alpha=0.0001, average=False, class_w
eight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='log', n_iter=5, n_jobs=1,
       penalty='l2', power_t=0.5, random_state=None, shuffle=True,
       verbose=0, warm...   penalty='l2', power_t=0.5, random_state=None, shuff
le=True,
       verbose=0, warm_start=False))])
```

In [20]:
```

Pipeline(steps=[('sgd_lr_1', SGDClassifier(alpha=0.0001, average=False, class_wei
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='log', n_iter=5, n_jobs=1,
       penalty='l2', power_t=0.5, random_state=None, shuffle=True,
       verbose=0, warm_start=False))])

yhat=pipeline.predict_proba(df)
```

```
/opt/conda/lib/python2.7/site-packages/sklearn/utils/deprecation.py:70: Depreca
tionWarning: Function transform is deprecated; Support to use estimators as fea
ture selectors will be removed in version 0.19. Use SelectFromModel instead.
  warnings.warn(msg, category=DeprecationWarning)
/opt/conda/lib/python2.7/site-packages/sklearn/linear_model/base.py:352: Runtim
eWarning: overflow encountered in exp
  np.exp(prob, prob)
```

In [21]:
```
def ensemble_predictions():
    return yhat[:,1]
```

In [22]:
```
def ensemble_predictions():
    return np.random.random(3000)

grader.score('ml__ensemble_model', ensemble_predictions)
```

```
==================
Your score:  0.41836160431
==================
```