LendingClub

Predicting Loan Defaults

Team Sparkling
Mengxin Qian, Will Young, April Wang, Jinxin Ma

Outline

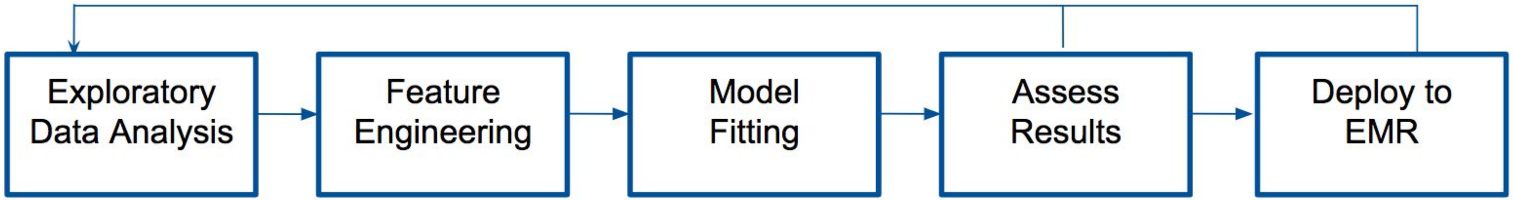
- Data & Introduction
- Feature Engineering
- Modeling
- Results and Evaluation
- Lessons Learned

LendingClub

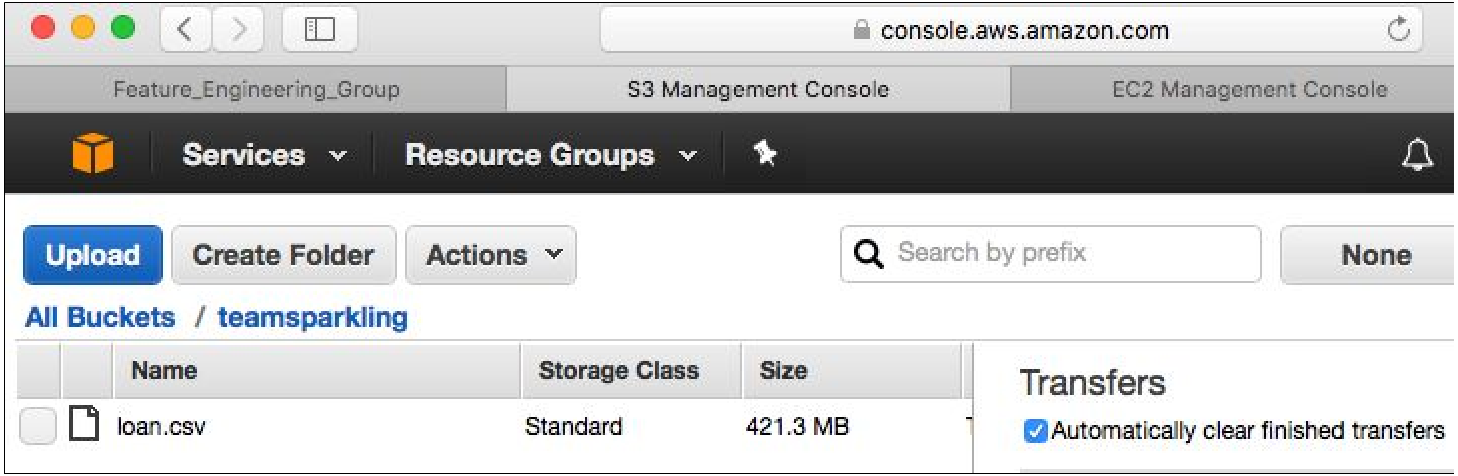
Data & Introduction

- Analytics goal
  - **Predict whether a customer will pay off the loan**
- Why we chose this data:
  - Interesting and challenging: 73 total features, 880,000 observations
  - Successful predictions increase profit for investors

Data processing



Data in S3 Bucket



Create Data Frame & Spark SQL

Not Secure https://ec2-35-167-233-46.us-west-2.compute.amazonaws.com:8888/notebooks/Featur

jupyter Feature_Engineering_Group (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Code CellToolbar

1. Import Libraries

In [1]:
```
from pyspark.sql.types import *
from pyspark.ml.feature import StringIndexer, OneHotEncoder
from pyspark.sql.functions import *
```

2. Load Data and Convert to Spark Data Frame

In [2]:
```
# load data as dataframe
loan_df = spark.read.csv("s3n://teamsparkling/loan.csv", header=True)
```

In [3]:
```
loan_df.rdd.getNumPartitions()
```

Out[3]: 64

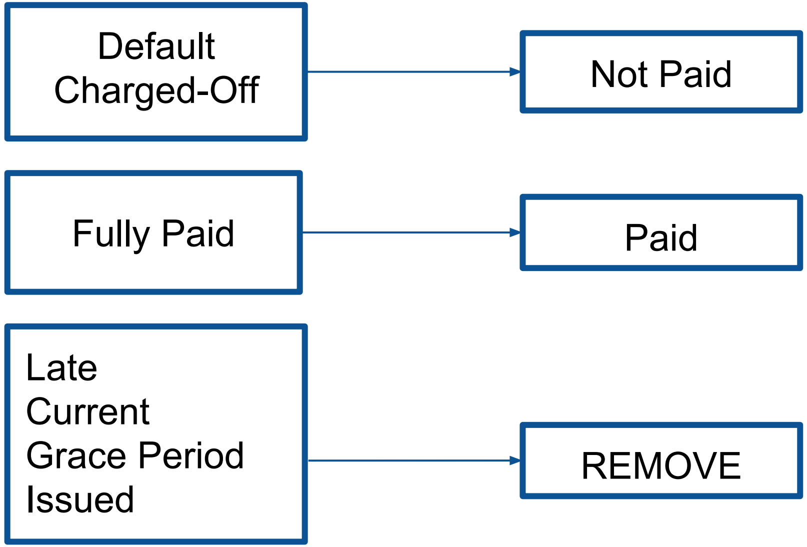
```
loan_df_final.select('paid_flag',
    'loan_amnt',
    'funded_amnt',
    'int_rate',
    'installment',
    'annual_inc').show(10)
```

```
+---------+---------+-----------+--------+-----------+----------+
|paid_flag|loan_amnt|funded_amnt|int_rate|installment|annual_inc|
+---------+---------+-----------+--------+-----------+----------+
|        0|   5000.0|     5000.0|   10.65|     162.87|   24000.0|
|        1|   2500.0|     2500.0|   15.27|      59.83|   30000.0|
|        0|   2400.0|     2400.0|   15.96|      84.33|   12252.0|
|        0|  10000.0|    10000.0|   13.49|     339.31|   49200.0|
|        0|   5000.0|     5000.0|     7.9|     156.46|   36000.0|
|        0|   3000.0|     3000.0|   18.64|     109.43|   48000.0|
|        1|   5600.0|     5600.0|   21.28|     152.39|   40000.0|
|        1|   5375.0|     5375.0|   12.69|     121.45|   15000.0|
|        0|   6500.0|     6500.0|   14.65|     153.45|   72000.0|
|        0|  12000.0|    12000.0|   12.69|     402.54|   75000.0|
+---------+---------+-----------+--------+-----------+----------+
only showing top 10 rows
```

Feature Engineering

- Create response variable
- Convert data type
- Bucket feature categories
- Encode categorical features
- Handle missing values

Feature Engineering - Response Variable



- *Current* does not imply the loan will be paid
- *Late* does not imply the loan will not be paid

Feature Engineering - Data Cleaning

```
+----------+
|emp_length|
+----------+
|   9 years|
|   5 years|
|    1 year|
|       n/a|
|   2 years|
|   7 years|
|   8 years|
|   4 years|
|   6 years|
|   3 years|
| 10+ years|
|  < 1 year|
+----------+
```

```
+-------+
|emp_len|
+-------+
|   null|
|      1|
|      6|
|      3|
|      5|
|      9|
|      4|
|      8|
|      7|
|     10|
|      2|
+-------+
```

```python
import re
def convert_to_int(s):
    s = re.sub('\\D', '', s)  #remove any non-digital character
    #\d matches any digital, #\D matches any non-digital
    try:
        return s
    except ValueError:
        return None

emp_to_num = udf(convert_to_int)
loan_df4 = loan_df3.withColumn('emp_len',emp_to_num('emp_length').cast('integer'))
```

Feature Engineering - Categorical Features

home_ownership
OWN
RENT
MORTGAGE
ANY
OTHER
NONE

re-group

home_ownership
OWN
Other
RENT
MORTGAGE

encode

home_ownership-onehot
(4,[1],[1.0])
(4,[1],[1.0])
(4,[1],[1.0])
(4,[1],[1.0])
(4,[1],[1.0])
(4,[1],[1.0])
(4,[2],[1.0])
(4,[1],[1.0])
(4,[2],[1.0])
(4,[2],[1.0])
(4,[1],[1.0])
(4,[1],[1.0])
(4,[1],[1.0])
(4,[1],[1.0])
(4,[1],[1.0])
(4,[0],[1.0])
(4,[0],[1.0])
(4,[1],[1.0])
(4,[1],[1.0])
(4,[1],[1.0])

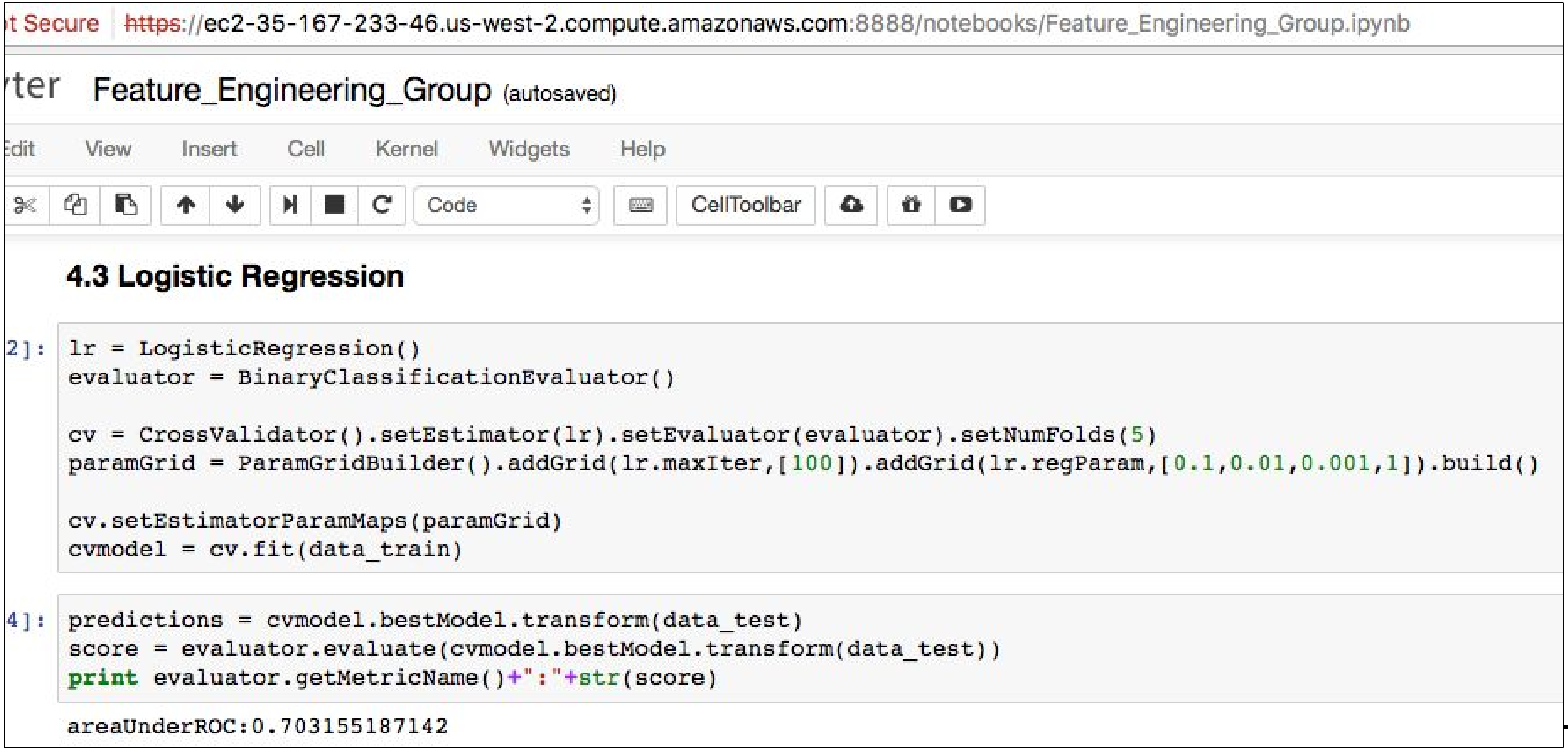
Feature Engineering - Missing Values

3.3.7 deal with missing values

```
# Filling NA values with 0
loan_df8 = loan_df8.fillna(0.0, ['tot_coll_amt','tot_cur_bal', 'total_rev_hi_lim'])
# Drop rows with NA
loan_df9 = loan_df8.dropna()
```

- Filling missing value with 0
- Drop rows with missing value

Logistic Regression & CV parameter tuning



t Secure https://ec2-35-167-233-46.us-west-2.compute.amazonaws.com:8888/notebooks/Feature_Engineering_Group.ipynb

ter Feature_Engineering_Group (autosaved)

Edit View Insert Cell Kernel Widgets Help

Code CellToolbar

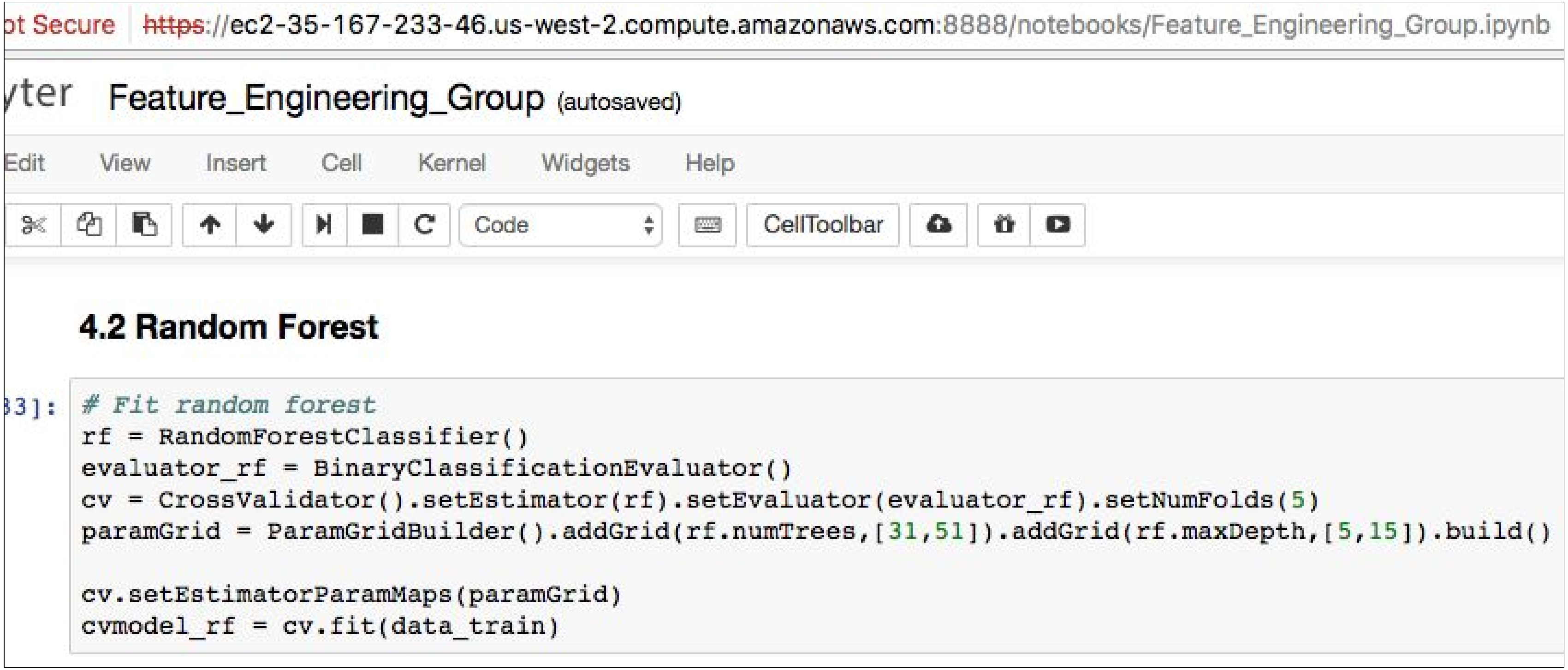
4.3 Logistic Regression

2]:
```
lr = LogisticRegression()
evaluator = BinaryClassificationEvaluator()

cv = CrossValidator().setEstimator(lr).setEvaluator(evaluator).setNumFolds(5)
paramGrid = ParamGridBuilder().addGrid(lr.maxIter,[100]).addGrid(lr.regParam,[0.1,0.01,0.001,1]).build()

cv.setEstimatorParamMaps(paramGrid)
cvmodel = cv.fit(data_train)
```

4]:
```
predictions = cvmodel.bestModel.transform(data_test)
score = evaluator.evaluate(cvmodel.bestModel.transform(data_test))
print evaluator.getMetricName()+":"+str(score)
```

```
areaUnderROC:0.703155187142
```

Random Forest & CV parameter tuning



ot Secure | https://ec2-35-167-233-46.us-west-2.compute.amazonaws.com:8888/notebooks/Feature_Engineering_Group.ipynb

yter Feature_Engineering_Group (autosaved)

Edit View Insert Cell Kernel Widgets Help

Code CellToolbar

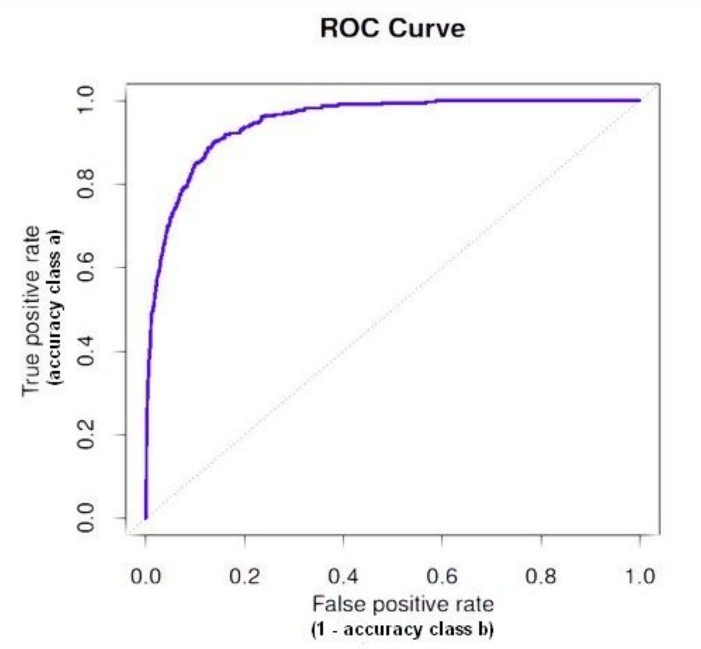
4.2 Random Forest

3]:

```
# Fit random forest
rf = RandomForestClassifier()
evaluator_rf = BinaryClassificationEvaluator()
cv = CrossValidator().setEstimator(rf).setEvaluator(evaluator_rf).setNumFolds(5)
paramGrid = ParamGridBuilder().addGrid(rf.numTrees,[31,51]).addGrid(rf.maxDepth,[5,15]).build()

cv.setEstimatorParamMaps(paramGrid)
cvmodel_rf = cv.fit(data_train)
```

Evaluation Metrics

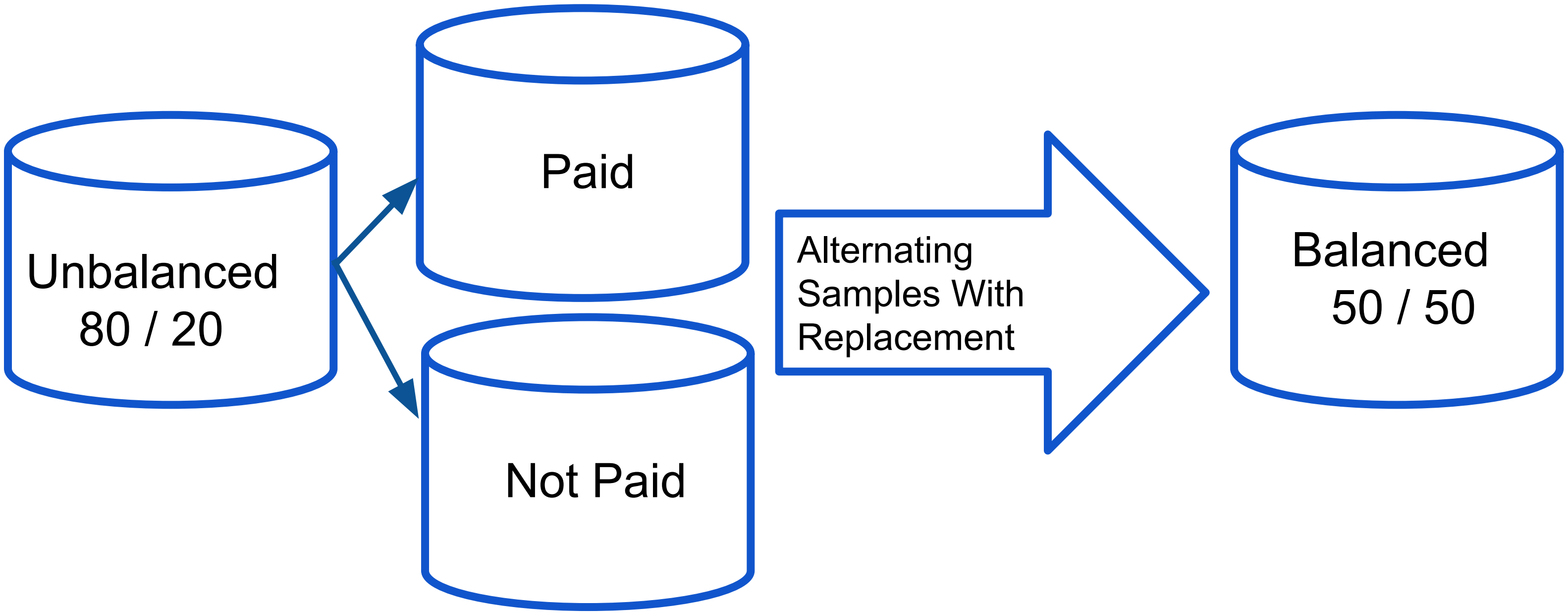
- areaUnderROC : 0.71
- Confusion matrix
  - Precision
  - Recall
  - Accuracy



Initial Test Results

	Random Forest
Precision	0.46
Recall	0.05
Accuracy	0.82

Balancing TRAIN Data



Test Results

	Random Forest (80/20 data)	Logistic Regression Model (50/50 data)	Random Forest Model (50/50 data)	Improvement
Precision	**0.46**	**0.29**	**0.28**	-37.23%
Recall	**0.05**	**0.66**	**0.65**	1215.09%
Accuracy	**0.82**	**0.64**	**0.64**	-21.48%

Test Results

```
metrics = MulticlassMetrics(sc.parallelize(predictionAndLabels))
print metrics.confusionMatrix().toArray()
print metrics.precision(1)
print metrics.recall(1)
print metrics.accuracy
```

```
[[ 25876.  14500.]
 [  3066.   5887.]]
0.288762446657
0.657544956998
0.64390115348
```

	Predicted Paid	Predicted Not Paid	Metrics
Actual Paid	**26,000 (53%)**	**14,500 (30%)**	Precision = 0.28
Actual Not Paid	**3,000 (5%)**	**6,000 (12%)**	Recall = 0.66

Lessons Learned

- Unbalanced training set can be problematic
- Recall is crucial in default detection
- Feature importance is hard to handle in ML
- Overhead issue with too many partitions

Thank you!