Not your parents' machine learning

Goodman Xiaoyuan Gu | Head of Marketing Data Engineering | Atlassian
Customer churns are very costly to any business - $$$ to acquire a replacement customer. Early warnings allow us to incentivize and engage with them to improve satisfaction and retention.
How can we improve activation rate from evaluator -> paying customer?
• **evaluator:** who are at risk of churning but worth attempting to save? who are predicted to retain but might swing?

• **behavior:** why those who stay and those who churn are different?

• **content:** what content resonates with evaluators?

• **engagement channel:** how to best engage with evaluators i.e. email, phone call, chat, push?

• **activation rate:** how does it change over the course of the 1st week, and what's driving it?
E2E PROCESS: CHURN PREDICTION UNLEASHED

- business objective
- user and use cases
- value proposition
- assumptions

- current solution
- baseline performance
- gaps and issues

- concept
- supervised/unsupervised/RL
- classification / regression
- online / batch learning
- multivariate / univariate
- single machine / distributed
- design considerations:
  - timeliness
  - scale of data
  - rate of change

- business metric
- format: confusion matrix, classification report
- performance metrics: precision, recall, F1 score, F2 score, accuracy...

- collect data
- prep data for ML: wrangling, data imputing, data scaling, train/test split, cross-validation
- feature engineering: discover and visualize data to gain insights, correlation study, principal component analysis (PCA), data quality assessment, derived features development
- build and train model
- refine model and tune hyper-parameters
- evaluate model with test data
- productionize, launch and monitor

Frame the problem
Gather status quo
Design the concept
Define success metrics
Build the solution

Define success metrics
THE REAL ISSUE

THE TRAINING DATA

- 90 days worth of product usage
- 57700 observations
- train/test split of 0.33
- data ingestion with SparkSQL jobs using EMR cluster, scheduled through Airflow
- stored on and served through AWS S3, and queryable through Athena
- re-training once/week
# convert to single precision to speed up
X = dataframe_features.values.astype(np.float32)
y = dataframe_target.values.astype(np.int32)

# drop features that are extremely sparse.
drop_list = ['instance',
             'eval_start_date',
             'retained',
             'watchers_added',
             'w1_active_users']
dataframe_features = raw_data.drop(drop_list,
                                     axis=1, inplace=False)

# scale/normalize the data
scaler = MaxAbsScaler()
X = scaler.fit_transform(X)

# transform X to fix missing data
imputer = Imputer(strategy='median')
imputed_x = imputer.fit_transform(X)
Logistic Regression Model:
Precision score: 0.90
Recall score: 0.47
Accuracy score: 0.86
Confusion matrix:
[[14296 239]
 [ 2405 2101]]
Classification report:

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.86</td>
<td>0.98</td>
<td>0.92</td>
</tr>
<tr>
<td>1</td>
<td>0.90</td>
<td>0.47</td>
<td>0.61</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.87</td>
<td>0.86</td>
<td>0.84</td>
</tr>
</tbody>
</table>

XGBoost Model:
Precision score: 0.80
Recall score: 0.64
Accuracy score: 0.88
Confusion matrix:
[[13814 721]
 [ 1636 2870]]
Classification report:

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.89</td>
<td>0.95</td>
<td>0.92</td>
</tr>
<tr>
<td>1</td>
<td>0.80</td>
<td>0.64</td>
<td>0.71</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.87</td>
<td>0.88</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Random Forest Classifier Model:
Precision score: 0.76
Recall score: 0.63
Accuracy score: 0.87
Confusion matrix:
[[13651 884]
 [ 1660 2846]]
Classification report:

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.89</td>
<td>0.94</td>
<td>0.91</td>
</tr>
<tr>
<td>1</td>
<td>0.76</td>
<td>0.63</td>
<td>0.69</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.86</td>
<td>0.87</td>
<td>0.86</td>
</tr>
</tbody>
</table>

MLP Classifier Model:
Precision score: 0.83
Recall score: 0.58
Accuracy score: 0.87
Confusion matrix:
[[14010 525]
 [ 1875 2631]]
Classification report:

<table>
<thead>
<tr>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.88</td>
<td>0.96</td>
<td>0.92</td>
</tr>
<tr>
<td>1</td>
<td>0.83</td>
<td>0.58</td>
<td>0.69</td>
</tr>
<tr>
<td>avg / total</td>
<td>0.87</td>
<td>0.87</td>
<td>0.87</td>
</tr>
</tbody>
</table>
“We may want to reconsider the tradeoff between spending time and money on algorithm development vs. spending it on corpus development”

- Michele Banko et al., Microsoft Research
- Peter Norvig et al., Google
CREATE EXTERNAL TABLE {marketing_schema}.instances_modeling (  instance INT ,eval_start_date STRING ,retained INT ,number_of_projects INT ,number_of_issues INT ,number_of_invites INT ,w1_active_users INT ,w1_agg_active_users INT ,w1_max_active_users INT ,watchers_added INT ,issues_updated INT ,issues_commented INT ,issues_assigned INT ,at_mentions INT ,issues_viewed INT ,issues_completed INT ,mobile_usage INT ,sprint_started INT ,sprint_finished INT )  ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat' OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat' LOCATION 's3://{s3_bucket_mgmt_de}/models/instances_modeling/v0' TBLPROPERTIES ('skip.header.line.count'='1');
Productionizing: Training Data Job

Job can be scheduled as a DAG in Airflow or entry in crontab

from pyspark.sql import SparkSession
from pyspark.sql.types import *
from etl_spark.util import read_text_file
import os

JOB_NAME = 'instances_modeling'
OUTPUT_S3_URI= os.path.join('s3://', S3_BUCKET_MGMT_DE, 'models',JOB_NAME,'v0')

spark = SparkSession.builder.master(spark_master).appName(JOB_NAME).enableHiveSupport().getOrCreate()

def run():
    spark.conf.set("spark.sql.parquet.binaryAsString", "true")
    sql = read_text_file(os.path.join(DIR_ETL_JOBS, JOB_NAME, 'instances_modeling.sql'))
    df = spark.sql(sql.format(marketing_schema=MARKETING_SCHEMA))
    df.coalesce(1).write.csv(path=OUTPUT_S3_URI, mode='overwrite', sep=';', header=True

Job can be scheduled as a DAG in Airflow or entry in crontab

Future data

Past data

Training

Training results

Prediction

Prediction results

① Training step

② Prediction step
Productionizing: Prediction Data Job

Job can be scheduled as a DAG in Airflow or entry in crontab, just more frequent

```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from etl_spark.util import read_text_file
import os

JOB_NAME = 'instances_w1'
OUTPUT_S3_URI = os.path.join('s3://', S3_BUCKET_MGMT_DE, 'models', JOB_NAME, 'v0')

spark = SparkSession.builder.master(spark_master).appName(JOB_NAME).enableHiveSupport().getOrCreate()

def run()
    spark.conf.set("spark.sql.parquet.binaryAsString","true")
    sql = read_text_file(os.path.join(DIR_ETL_JOBS, JOB_NAME, 'instances_w1.sql'))
    df = spark.sql(sql.format(marketing_schema=MARKETING_SCHEMA))
    df.coalesce(1).write.csv(path=OUTPUT_S3_URI, mode='overwrite', sep=',', header=True
```

Future data

Past data  Training results  Prediction results

① Training step

② Prediction step

Job can be scheduled as a DAG in Airflow or entry in crontab, just more frequent
Productionizing: Model Training and Prediction Jobs

Jobs can be scheduled as a DAG in Airflow or entry in crontab on production EC2 insurance/EMR Cluster

```
#!/bin/bash

# Training step
echo "start the virtual env"
export VIRTUAL_ENV_PATH=/opt/virtualenvs
PP_VENV=${VIRTUAL_ENV_PATH}/propensity-prediction-venv
source ${PP_VENV}/bin/activate

# Run the propensity prediction model.py
export PP_HOME=/opt/mgmt/propensity_prediction/ep
cd ${PP_HOME}
python ${PP_HOME}/model.py

decho "deactivate the virtual env"
deactivate

```

```
#!/bin/bash

# Prediction step

echo "start the virtual env"
export VIRTUAL_ENV_PATH=/opt/virtualenvs
PP_VENV=${VIRTUAL_ENV_PATH}/propensity-prediction-venv
source ${PP_VENV}/bin/activate

# Run the propensity prediction predict.py
export PP_HOME=/opt/mgmt/propensity_prediction/ep
cd ${PP_HOME}
python ${PP_HOME}/predict.py

decho "deactivate the virtual env"
deactivate
```
Training

Single algorithm used by ~60% Kaggle Competition winning teams

**Extreme Gradient Boosting**
- Sparse-aware implementation fixing missing data
- Block Structure for parallel tree construction
- Parallelization using CPU cores during training
- Distributed Computing for large models
- Out-of-Core Computing for very large datasets that don’t fit into memory
- Cache Optimization of data structures and algorithm
- Continued Training - boost fitted model on new data

from xgboost import XGBClassifier

# data prep and feature engineering

# with tuned hyperparameters
model = XGBClassifier(
    learning_rate=0.1,
    n_estimators=200,
    max_depth=3,
    min_child_weight = 6,
    gamma = 0,
    subsample=0.5,
    colsample_bytree=1.0,
    colsample_bylevel=1.0,
    objective='binary:logistic',
    nthread=-1,
    scale_pos_weight = 1,
    seed=27)

# train the model
model.fit(X_train, y_train)

# make predictions
predictions = model.predict(X_test)

# evaluate with test set

# persist model
joblib.dump(model, MODEL_PATH)
s3_r.meta.client.upload_file(MODEL_PATH, Bucket=BUCKET, Key=MODEL_PATH_REMOTE)
from xgboost import XGBClassifier

obj = s3.get_object(Bucket=BUCKET,
    Key=objs['Contents'][-1]['Key'])

# load prediction data
data_frame =
pd.read_csv(io.BytesIO(obj['Body'].read()))

s3_model.meta.client.download_file(Bucket=
    BUCKET, Key=MODEL_PATH_REMOTE, Filename=MODEL_PATH)

# load persisted XGBoost model
predictor = joblib.load(MODEL_PATH)

# feature selection

# scale the values of selected features
scaler = MaxAbsScaler()
features_scaled = scaler.fit_transform(features_selected)

# transform features
imputer = Imputer(strategy='median')
imputed_x = imputer.fit_transform(features_selected)

# make predictions
new_predictions = predictor.predict(imputed_x)

# add predictions as a new column to the original data frame
data_frame['prediction_retained'] = new_predictions

new_data.to_csv(LOCAL_FILE_PATH, index=False)
s3_r.meta.client.upload_file(LOCAL_FILE_PATH, Bucket=BUCKET,
    Key=FILE_PATH)

---

XGBoost

• Single algorithm used by ~60% Kaggle Competition winning teams
• **Extreme Gradient Boosting**
  • superior overall performance
  • excellent execution speed
  • relatively small footprint
  • easy model persistency
What are some challenges you can imagine?
AMAZON SAGEMAKER

- managed service - easily build, train, and deploy machine learning models
- hosted Jupyter notebooks - explore and visualize training data
- 12 algorithms pre-installed and optimized
- pre-configured to run TensorFlow and Apache MXNet
- single-click training in the console or with a simple API call
- automated Hyperparameter Optimization (HPO)
- deploys model on cluster for performance and availability
- built-in A/B testing capabilities for experiments
- easy to integrate machine learning models into applications by providing an HTTPS endpoint

★ complexity transparency
★ faster time to market
★ tight integration with existing data workflow
Workflow Demo of Churn Prediction with Sagemaker
We have gone through this

- Local Machine
- Cloud EC2 Instances
- Amazon SageMaker
- Cloud ML Platform
We will go build

- Churn Prediction Unleashed (CPU)
- Generic Prediction Utility (GPU)
- Application Specific Inference Capability (ASIC)
We are hiring...