Cloud AI Platform + What-if Tool: end-to-end XGBoost example

This notebook shows how to:

- Build a binary classification model with XGBoost trained on a mortgage dataset
- Deploy the model to Cloud AI Platform
- Use the What-if Tool on your deployed model

In [ ]:

```
#You'll need to install XGBoost on the TF instance
!pip3 install xgboost==0.90 witwidget --user --quiet
```

After doing a pip install, restart your kernel by selecting kernel from the menu and clicking Restart Kernel before proceeding further

In [1]:

```
import pandas as pd
import xgboost as xgb
import numpy as np
import collections
import witwidget

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.utils import shuffle
from witwidget.notebook.visualization import WitWidget, WitConfigBuilder
```

Download and pre-process data

In this section we'll:

- Download a subset of the mortgage dataset from Google Cloud Storage
- Because XGBoost requires all columns to be numerical, we'll convert all categorical columns to dummy columns (0 or 1 values for each possible category value)
- Note that we've already done some pre-processing on the original dataset to convert value codes to strings: for example, an agency code of `1` becomes `Office of the Comptroller of the Currency (OCC)`

In [2]:

```
# Use a small subset of the data since the original dataset is too big for Colab (2.5GB)
# Data source: https://www.ffiec.gov/hmda/hmdaflat.htm
!gsutil cp gs://mortgage_dataset_files/mortgage-small.csv .
```

```
Copying gs://mortgage_dataset_files/mortgage-small.csv...
| [1 files][330.8 MiB/330.8 MiB]
Operation completed over 1 objects/330.8 MiB.
```

In [3]:

```
# Set column dtypes for Pandas
COLUMN_NAMES = collections.OrderedDict({
  'as_of_year': np.int16,
  'agency_code': 'category',
  'loan_type': 'category',
  'property_type': 'category',
  'loan_purpose': 'category',
  'occupancy': np.int8,
  'loan_amt_thousands': np.float64,
  'preapproval': 'category',
  'county_code': np.float64,
  'applicant_income_thousands': np.float64,
  'purchaser_type': 'category',
  'hoepa_status': 'category',
  'lien_status': 'category',
  'population': np.float64,
  'ffiec_median_fam_income': np.float64,
  'tract_to_msa_income_pct': np.float64,
  'num_owner_occupied_units': np.float64,
  'num_1_to_4_family_units': np.float64,
  'approved': np.int8
})
```

In [5]:

```
# Load data into Pandas
data = pd.read_csv(
  'mortgage-small.csv',
  index_col=False,
  dtype=COLUMN_NAMES
)
data = data.dropna()
data = shuffle(data, random_state=2)
data.head()
```

Out[5]:

	as_of_year	agency_code	loan_type	property_type	loan_purpose	occupancy	loan_amt_thousands	preappro
310650	2016	Consumer Financial Protection Bureau (CFPB)	Conventional (any loan other than FHA, VA, FSA...	One to four-family (other than manufactured ho...	Refinancing	1	110.0	applica
630129	2016	Department of Housing and Urban Development (HUD)	Conventional (any loan other than FHA, VA, FSA...	One to four-family (other than manufactured ho...	Home purchase	1	480.0	applica
715484	2016	Federal Deposit Insurance Corporation (FDIC)	Conventional (any loan other than FHA, VA, FSA...	One to four-family (other than manufactured ho...	Refinancing	2	240.0	applica
887708	2016	Office of the Comptroller of the Currency (OCC)	Conventional (any loan other than FHA, VA, FSA...	One to four-family (other than manufactured ho...	Refinancing	1	76.0	applica
719598	2016	National Credit Union Administration (NCUA)	Conventional (any loan other than FHA, VA, FSA...	One to four-family (other than manufactured ho...	Refinancing	1	100.0	applica

In [6]:

```
# Label preprocessing
labels = data['approved'].values

# See the distribution of approved / denied classes (0: denied, 1: approved)
print(data['approved'].value_counts())
```

```
1    665389
```

```
0    334610
Name: approved, dtype: int64
```

In [7]:

```
data = data.drop(columns=['approved'])
```

In [8]:

```
# Convert categorical columns to dummy columns
dummy_columns = list(data.dtypes[data.dtypes == 'category'].index)
data = pd.get_dummies(data, columns=dummy_columns)
```

In [9]:

```
# Preview the data
data.head()
```

Out[9]:

	as_of_year	occupancy	loan_amt_thousands	county_code	applicant_income_thousands	population	ffiec_mediar
310650	2016	1	110.0	119.0	55.0	5930.0	
630129	2016	1	480.0	33.0	270.0	4791.0	
715484	2016	2	240.0	59.0	96.0	3439.0	
887708	2016	1	76.0	65.0	85.0	3952.0	
719598	2016	1	100.0	127.0	70.0	2422.0	

5 rows × 44 columns

Train the XGBoost model

In [10]:

```
# Split the data into train / test sets
x,y = data,labels
x_train,x_test,y_train,y_test = train_test_split(x,y)
```

In [11]:

```
# Train the model, this will take a few minutes to run
bst = xgb.XGBClassifier(
    objective='reg:logistic'
)

bst.fit(x_train, y_train)
```

```
/home/jupyter/.local/lib/python3.7/site-packages/xgboost/sklearn.py:1146: UserWarning: The u
se of label encoder in XGBClassifier is deprecated and will be removed in a future release.
To remove this warning, do the following: 1) Pass option use_label_encoder=False when constr
ucting XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e.
0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

Out[11]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=4, num_parallel_tree=1,
              objective='reg:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [12]:

```
# Get predictions on the test set and print the accuracy score
y_pred = bst.predict(x_test)
acc = accuracy_score(y_test, y_pred.round())
print(acc, '\n')
```

```
0.873392
```

In [13]:
```
# Print a confusion matrix
print('Confusion matrix:')
cm = confusion_matrix(y_test, y_pred.round())
cm = cm / cm.astype(np.float).sum(axis=1)
print(cm)
```

```
Confusion matrix:
[[0.85184475 0.07427336]
 [0.23100144 0.88419409]]
```

In [14]:
```
# Save the model so we can deploy it
bst.save_model('model.bst')
```

Deploy model to AI Platform

Copy your saved model file to Cloud Storage and deploy the model to AI Platform. In order for this to work, you'll need the Cloud AI Platform Models API enabled. Update the values in the next cell with the info for your GCP project. Replace GCP_PROJECT with the value in the Qwiklabs lab page for GCP Project ID in the left pane, replace MODEL_BUCKET with gs:// with the value for BucketName appended, and replace MODEL_NAME with a name for your model.

In [18]:
```
GCP_PROJECT = 'qwiklabs-gcp-02-d94ebaeed409 '
MODEL_BUCKET = 'gs://gcp-02-d94ebaeed409'
MODEL_NAME = 'XGBoost_mortage_small' # You'll create this model below
VERSION_NAME = 'v1'
```

In [19]:
```
# Copy your model file to Cloud Storage
!gsutil cp ./model.bst $MODEL_BUCKET
```

```
Copying file://./model.bst [Content-Type=application/octet-stream]...
/ [1 files][291.5 KiB/291.5 KiB]
Operation completed over 1 objects/291.5 KiB.
```

In [20]:
```
# Configure gcloud to use your project
!gcloud config set project $GCP_PROJECT
```

```
Updated property [core/project].
```

In [21]:
```
# Create a model
!gcloud ai-platform models create $MODEL_NAME --regions us-central1
```

```
Using endpoint [https://ml.googleapis.com/]
Created ai platform model [projects/qwiklabs-gcp-02-d94ebaeed409/models/XGBoost_mortage_smal
l].
```

In [22]:
```
# Create a version, this will take ~2 minutes to deploy
!gcloud ai-platform versions create $VERSION_NAME \
--model=$MODEL_NAME \
--framework='XGBOOST' \
--runtime-version=1.15 \
--origin=$MODEL_BUCKET \
--staging-bucket=$MODEL_BUCKET \
--python-version=3.7 \
--project=$GCP_PROJECT \
--region=global
```

```
Using endpoint [https://ml.googleapis.com/]
Creating version (this might take a few minutes)......done.
```

Using the What-if Tool to interpret your model

Once your model has deployed, you're ready to connect it to the What-if Tool using the `WitWidget`. **Note**: You can ignore the message `TypeError(unsupported operand type(s) for -: 'int' and 'list')` while

In [23]:
```
# Format a subset of the test data to send to the What-if Tool for visualization
# Append ground truth label value to training data

# This is the number of examples you want to display in the What-if Tool
num_wit_examples = 500
test_examples = np.hstack((x_test[:num_wit_examples].values,y_test[:num_wit_examples].reshap
```

In [24]:
```
# Create a What-if Tool visualization, it may take a minute to load
# See the cell below this for exploration ideas

# This prediction adjustment function is needed as this xgboost model's
# prediction returns just a score for the positive class of the binary
# classification, whereas the What-If Tool expects a list of scores for each
# class (in this case, both the negative class and the positive class).
def adjust_prediction(pred):
  return [1 - pred, pred]

config_builder = (WitConfigBuilder(test_examples.tolist(), data.columns.tolist() + ['mortgag
  .set_ai_platform_model(GCP_PROJECT, MODEL_NAME, VERSION_NAME, adjust_prediction=adjust_pre
  .set_target_feature('mortgage_status')
  .set_label_vocab(['denied', 'approved']))
WitWidget(config_builder, height=800)
```

What-if Tool exploration ideas

- **Individual data points**: the default graph shows all data points from the test set, colored by their ground truth label (approved or denied)
  - Try selecting data points close to the middle and tweaking some of their feature values. Then run inference again to see if the model prediction changes
  - Select a data point and then select the "Show nearest counterfactual datapoint" radio button. This will highlight a data point with feature values closest to your original one, but with the opposite prediction
- **Binning data**: create separate graphs for individual features
  - From the "Binning - X axis" dropdown, try selecting one of the agency codes, for example "Department of Housing and Urban Development (HUD)". This will create 2 separate graphs, one for loan applications from the HUD (graph labeled 1), and one for all other agencies (graph labeled 0). This shows us that loans from this agency are more likely to be denied
- **Exploring overall performance**: Click on the "Performance & Fairness" tab to view overall performance statistics on the model's results on the provided dataset, including confusion matrices, PR curves, and ROC curves.
  - Experiment with the threshold slider, raising and lowering the positive classification score the model needs to return before it decides to predict "approved" for the loan, and see how it changes accuracy, false positives, and false negatives.
  - On the left side "Slice by" menu, select "loan_purpose_Home purchase". You'll now see performance on the two subsets of your data: the "0" slice shows when the loan is not for a home purchase, and the "1" slice is for when the loan is for a home purchase. Check out the accuracy, false postive, and false negative rate between the two slices to look for differences in performance. If you expand the rows to look at the confusion matrices, you can see that the model predicts "approved" more often for home purchase loans.
  - You can use the optimization buttons on the left side to have the tool auto-select different positive classification thresholds for each slice in order to achieve different goals. If you select the "Demographic parity" button, then the two thresholds will be adjusted so that the model predicts "approved" for a similar percentage of applicants in both slices. What does this do to the accuracy, false positives and false negatives for each slice?