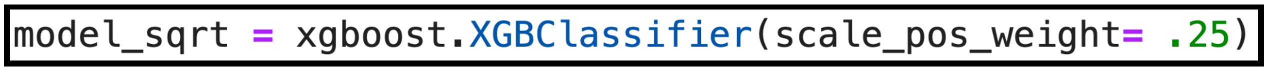
Bank Lending Prediction Task
Name: Gurney Halleck
Top 20%

Dataset + Task Description:

With 1 million rows each representing a person and 25 columns representing their respective financial features, I plan to use this dataset to predict if an individual will pay off their loan or not given the same features. The last column in the training data('loan_paid') acts as our predefined labels which we'll use to train the model.

Model:

Because it's a classification(binary) problem, I used XGBoost, a tree based ensemble learning algorithm, instead of XGBRegressor.

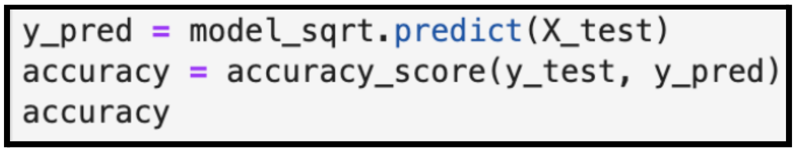


```
model_sqrt = xgboost.XGBClassifier(scale_pos_weight= .25)
```

One parameter I made sure to specify was the 'scale_pos_weight'. It helps with imbalanced datasets by specifying the ratio of negative classes to positive classes. In our test data, there were 20,000 instances of people not paying their loans and 80,000 instances of people who did pay them off, so I just used that ratio. I trained the model on a sample of 750,000 rows(75% of our data). Our output would be the test data's 250,000 predictions(either 1's or 0's) since our test data had 250,000 rows after splitting.

Scoring:

To score the models I generated, I used sk-learn's built-in .score functionality. The parameters it took in were our test data of 250,000 rows and our respective predictions. Then it evaluated how our model performed and gave us an accuracy.



```
y_pred = model_sqrt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Search Function:

A gradient measures the change in all weights with regard to the change in error. The larger the gradient the faster our model learns. Gradient boosting is a special case of boosting where misclassification/errors end up being minimized via gradient descent - which is an optimization algorithm that attempts to find a local or global minimum of a loss function (e.g. sum of squared error over the training set). XGBoost, in particular, optimizes standard Gradient Boosting Machines by doing things like parallelized tree building (building sequential trees), tree pruning (pruning trees backward using the 'max_depth' parameter), and regularization to prevent overfitting (penalizes complex models using both L1 and L2 regularization).

Fake Review Competition

Name: Gurney Halleck

Bottom 50%

Dataset + Task Description:

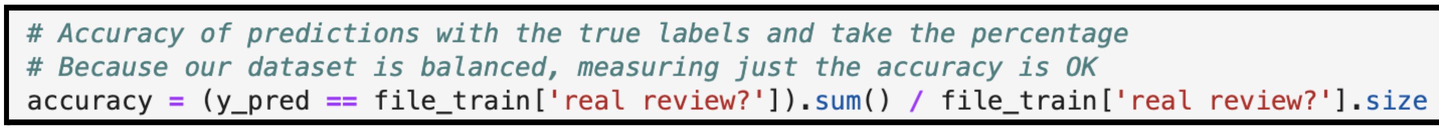
With 2249 rows each representing a person and 4 columns representing their respective financial features, I plan to use this dataset to predict if an individual will pay off their loan or not given the same features. The data consists of a train set,validation set, and test set where the train and validation dataset are identically distributed. The five features of the data frame are the review's ID, the item's category(e.g. 'Books'), it's rating out of 5 stars, the actual text as a string from the review, and the review's label which is 1 if it was real and 0 otherwise. We want to be able to predict the label of a review given the same features. The output we want is a .csv with one ID column and one probability column.

Model + Model Space:

I used logistic regression as my model for this competition because that's what the professor recommended we use. With the Logistic Regression model, I tested various inverse regularization parameters to see which performed the best. I tried C values from .001 to 100 and found that the lambda regulator is stronger while C gets lower and C equals 1/ƛ. Changing the values of C is crucial because we're able to see the disincentivization and regulation in place from overfitting. I changed the regularization parameter from Lasso to Ridge to reduce the overall model from chances of overfitting and to keep the word feature magnitudes small.

Score Function:

To score the models I generated, I just calculated the accuracy with the professor's starter code from homework 4.



```
# Accuracy of predictions with the true labels and take the percentage
# Because our dataset is balanced, measuring just the accuracy is OK
accuracy = (y_pred == file_train['real review?']).sum() / file_train['real review?'].size
```

Search Function:

The search function I used for my final model was a gradient descent with cross-entropy. I attempted to minimize the negative log probability values and get the true y distribution as accurately as possible compared to our predicted y distribution. I then used boosting after looking at the built in features and receiving the cross-entropy loss for each model. Subsequently, the base classifier logistic regression model improved with the new recalculated weights.