# CS 395
# Homework 7 – Training Boosted Trees models in TensorFlow
**25 Points Total**
**Due in Canvas by 11:59 PM on Sunday, April 7, 2019**

This lab has us training a Gradient Boosting model using decision trees with the tf.estimator API.

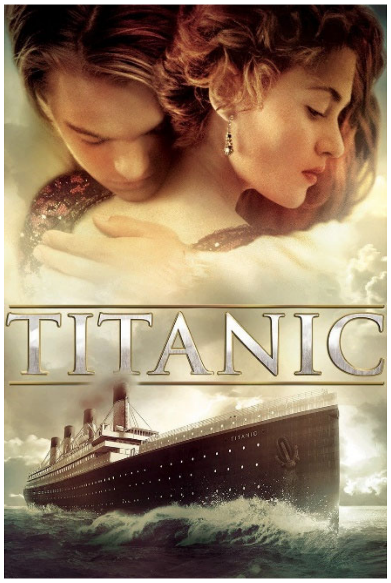
Boosted Trees models are among the most popular and effective machine learning approaches for both regression and classification. It is an ensemble technique that combines the predictions from several (think 10s, 100s or even 1000s) tree models.

Boosted Trees models are popular with many data scientists as they can achieve impressive performance with minimal hyper parameter tuning.

**Data Preparation**

You will be using the titanic dataset, where the (rather morbid) goal is to predict passenger survival, given characteristics such as gender, age, class, etc.

Cue up the Celine Deon music….



**Note**: In the code below, replace the number 123 with the last 3 digits of your Bear Number as the random seed. Run the following.

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

tf.enable_eager_execution()

tf.logging.set_verbosity(tf.logging.ERROR)
tf.set_random_seed(123)
```

```
# Load dataset.
dftrain =
pd.read_csv('https://storage.googleapis.com/tfbt/titanic_train.csv')
dfeval =
pd.read_csv('https://storage.googleapis.com/tfbt/titanic_eval.csv')
y_train = dftrain.pop('survived')
y_eval = dfeval.pop('survived')
```

The dataset consists of a training set and an evaluation set:

- dftrain and y_train are the training set—the data the model uses to learn.
- The model is tested against the eval set, dfeval, and y_eval.

**Explore the Data**

Now let's take a look at the data.

1. Run the 4 commands below and show the output for each (1 point)

```
dftrain.head()

dftrain.describe()

dfeval.head()

dfeval.describe()
```

Now let's look at some additional information about our data. Run the following, which will show the shape of our dataset:

```
dftrain.shape[0], dfeval.shape[0]
```

2. Show the output. What happens if we change the array element in the above arrays from '0' to other values? What information does this value provide? (1 point)

Now let's look at some other plotted information about our data.

```
dftrain.age.hist(bins=20);

dftrain.sex.value_counts().plot(kind='barh');
```

3. What information can you ascertain about the age and gender of passengers in your training dataset? (1 point)

4. Now run the above commands for the evaluation dataset. What information can you ascertain about the age and gender of passengers in your evaluation (test) dataset? (1 point)

Run the following:

```
(dftrain['class']
  .value_counts()
  .plot(kind='barh'));
```

Most passengers were in third class, just like Jack Dawson (Leonardo DiCaprio)

```
(dftrain['embark_town']
  .value_counts()
  .plot(kind='barh'));
```

Most boarded in Southampton.

5. Run the above and show your code and the output. For the third one, change 'class' to another *meaningful* feature or attribute (besides 'embark_town') and show the output. Describe what information this provides. (1 point)

**Understanding the Predictive Model**

6. Run the following and show your output (1 point):

```
ax = (pd.concat([dftrain, y_train], axis=1)\
  .groupby('sex')
  .survived
  .mean()
  .plot(kind='barh'))
ax.set_xlabel('% survive');
```

Much to the benefit of Rose DeWitt Bukater (Kate Winslet), but to the detriment of her fiancé Cal Hockley (Billy Zane) females have a much higher chance of surviving vs. males. This will clearly be a predictive feature for our model.

7. Change the above code and run it to see the chance of survival by **class** and show the output. What is your finding? (2 points).

**Create feature columns and input functions**

The Gradient Boosting estimator can utilize both numeric and categorical features. Feature columns work with all TensorFlow estimators and their purpose is to define the features used for modeling. Additionally they provide some feature engineering capabilities like one-hot-encoding, normalization, and bucketization.

In this lab, the fields in CATEGORICAL_COLUMNS are transformed from categorical columns to one-hot-encoded columns (indicator column). Run the following:

```
fc = tf.feature_column
CATEGORICAL_COLUMNS = ['sex', 'n_siblings_spouses', 'parch', 'class', 'deck',
                       'embark_town', 'alone']
```

```
NUMERIC_COLUMNS = ['age', 'fare']

def one_hot_cat_column(feature_name, vocab):
  return fc.indicator_column(
      fc.categorical_column_with_vocabulary_list(feature_name,
                                                 vocab))
feature_columns = []
for feature_name in CATEGORICAL_COLUMNS:
  # Need to one-hot encode categorical features.
  vocabulary = dftrain[feature_name].unique()
  feature_columns.append(one_hot_cat_column(feature_name, vocabulary))

for feature_name in NUMERIC_COLUMNS:
  feature_columns.append(fc.numeric_column(feature_name,
                                           dtype=tf.float32))
```

We can view the transformation that a feature column produces. For example, here is the output when using the indicator_column on a single example:

8. Now run the code below and show the output. Why do we need to one-hot encode the class? (1 point)

```
example = dftrain.head(1)
class_fc = one_hot_cat_column('class', ('First', 'Second', 'Third'))
print('Feature value: "{}"'.format(example['class'].iloc[0]))
print('One-hot encoded: ', fc.input_layer(dict(example), [class_fc]).numpy())
```

Additionally, you can view all of the feature column transformations together.

```
      fc.input_layer(dict(example), feature_columns).numpy()
```

9. Run the above, show the output, and explain what information it is providing us regarding this one-hot encoding (1 point).

Next you need to create the input functions. These will specify how data will be read into our model for both training and evaluation (test). You will use the from_tensor_slices method in the tf.data API to read in data directly from Pandas. This is suitable for smaller, in-memory datasets. For larger datasets, the tf.data API supports a variety of file formats (including csv) so that you can process datasets that do not fit in memory. First we use a linear classified (given in bold below). Run the following.

```
# Use entire batch since this is such a small dataset.
NUM_EXAMPLES = len(y_train)

def make_input_fn(X, y, n_epochs=None, shuffle=True):
  def input_fn():
    dataset = tf.data.Dataset.from_tensor_slices((dict(X), y))
    if shuffle:
      dataset = dataset.shuffle(NUM_EXAMPLES)
    # For training, cycle thru dataset as many times as need (n_epochs=None).
    dataset = dataset.repeat(n_epochs)
    # In memory training doesn't use batching.
    dataset = dataset.batch(NUM_EXAMPLES)
    return dataset
  return input_fn

# Training and evaluation input functions.
train_input_fn = make_input_fn(dftrain, y_train)
eval_input_fn = make_input_fn(dfeval, y_eval, shuffle=False, n_epochs=1)
```

**Train and evaluate the model**

Below you will do the following steps:

- Initialize the model, specifying the features and hyperparameters.
- Feed the training data to the model using the train_input_fn and train the model using the train function.
- You will assess model performance using the evaluation set—in this example, the dfeval DataFrame. You will verify that the predictions match the labels from the y_eval array.

Before training a Boosted Trees model, let's first train a linear classifier (logistic regression model). It is best practice to start with simpler model to establish a benchmark.

```
linear_est = tf.estimator.LinearClassifier(feature_columns)

# Train model.
linear_est.train(train_input_fn, max_steps=100)

# Evaluation.
results = linear_est.evaluate(eval_input_fn)
print('Accuracy : ', results['accuracy'])
print('Dummy model: ', results['accuracy_baseline'])
```

10. Run the above and show your output. Describe what the linear classifier is doing (2 points).

Now run the following with a different classifier.

Let's train a Boosted Trees model. For boosted trees, regression (BoostedTreesRegressor) and classification (BoostedTreesClassifier) are supported, along with using any twice differentiable custom loss (BoostedTreesEstimator).

Since the goal is to predict a class - survive or not survive, you will use the BoostedTreesClassifier.

```
# Since data fits into memory, use entire dataset per layer. It will be
faster.
# Above one batch is defined as the entire dataset.
n_batches = 1
est = tf.estimator.BoostedTreesClassifier(feature_columns,
                                          n_batches_per_layer=n_batches)

# The model will stop training once the specified number of trees is built,
not
# based on the number of steps.
est.train(train_input_fn, max_steps=100)

# Eval.
```

```
results = est.evaluate(eval_input_fn)
print('Accuracy : ', results['accuracy'])
print('Dummy model: ', results['accuracy_baseline'])
```

11. Run the above and show your code and output. Describe what the boosted trees classifier is doing. (2 points).
12. What happens if we increase the number of batches beyond 1? Change the value in the above code and show the code and output. Why did this value change? (2 points)
13. Explain how the boosted tree classifier differs from the linear classifier (you will need to do some research here) (2 points).

Let's look at the frequency counts of the likeliness of survival.

14. Run the following and show code and your output (1 point).

```
pred_dicts = list(est.predict(eval_input_fn))
probs = pd.Series([pred['probabilities'][1] for pred in pred_dicts])

probs.plot(kind='hist', bins=20, title='predicted probabilities');
```

15. Provide an explanation of the output. (2 points)

Last, we want to examine a ROC curve. A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

- The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning.

- The false-positive rate is also known as the fall-out or probability of false alarm and can be calculated as (1 – specificity).

It can also be thought of as a plot of the power as a function of the Type I Error of the decision rule (when the performance is calculated from just a sample of the population, it can be thought of as estimators of these quantities). The ROC curve is thus the sensitivity as a function of fall-out.

```
from sklearn.metrics import roc_curve
from matplotlib import pyplot as plt

fpr, tpr, _ = roc_curve(y_eval, probs)
plt.plot(fpr, tpr)
plt.title('ROC curve')
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.xlim(0,)
plt.ylim(0,);
```

16. Run the above and show the code and output. (1 point).

17. What general information does the ROC curve provide about the survival rate of the Titanic passengers? In other words, provide an analysis of the ROC curve information provided here toward our prediction model. (3 points).