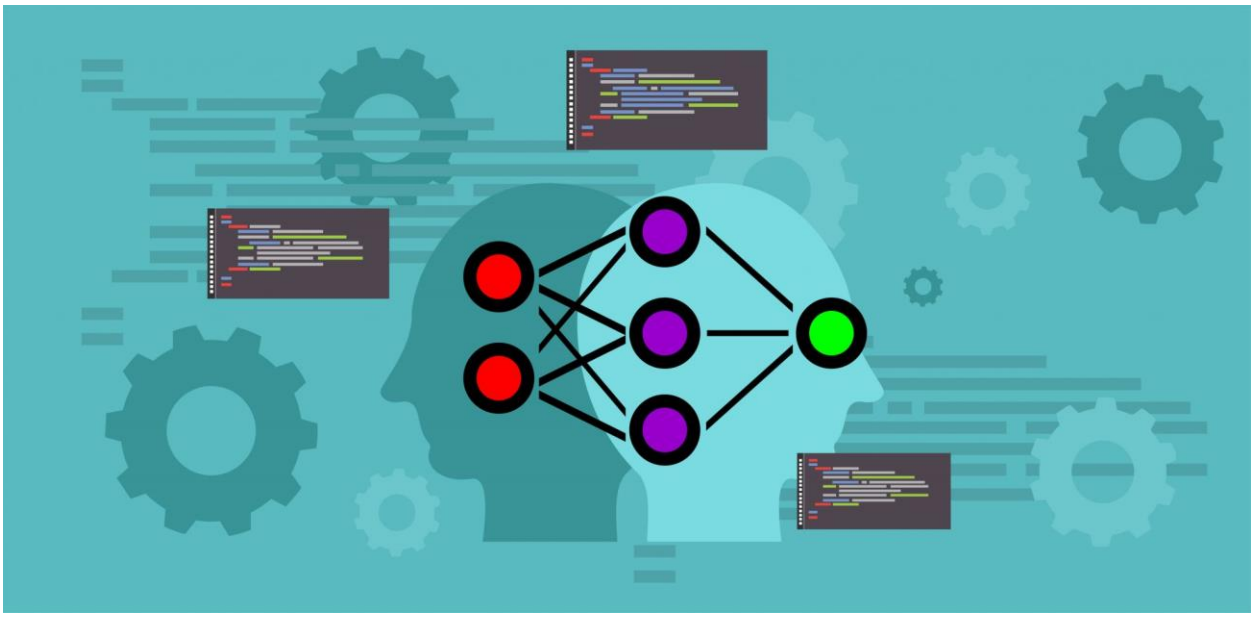
Implementing Classification Algorithms on Donors Choose Data



Chapter 1

Introduction

1.1 Problem Statement:

DonorsChoose.org has funded over 1.1 million classroom requests through the support of 3 million donors, the majority of whom were making their first-ever donation to a public school. If DonorsChoose.org can motivate even a fraction of those donors to make another donation, that could have a huge impact on the number of classroom requests fulfilled.

The dataset contains project proposals received over a period of time for classroom projects in need of funding. Using the text of project descriptions as well as additional metadata about the project, teacher, and school. Classification algorithms are implemented on these feature sets to find out the best hyper parameter which will give us maximum AUC value and plot the performance of the model.

1.2 Data:

Features that are needed to be considered, Dataset shape (109248,8)

Feature	Description
project_id	A unique identifier for the proposed project. **Example:** p036502
project_grade_category	Grade level of students for which the project is targeted. One of the following enumerated values: • Grades PreK-2 • Grades 3-5 • Grades 6-8

project_subject_categories	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • Applied Learning • Care & Hunger • Health & Sports • History & Civics • Literacy & Language • Math & Science • Music & The Arts
school_state	State where school is located (Two-letter U.S. postal code). **Example:** WY
project_subject_subcategories	One or more (comma-separated) subject subcategories for the project. **Examples:** • Literacy • Literature & Writing, Social Sciences
teacher_prefix	Teacher's title. One of the following enumerated values: • nan • Dr. • Mr. • Mrs. • Ms. • Teacher.
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same teacher. **Example:** 2
price	Price of the resource required. **Example:** 9.95
Project essay	Application essay

Chapter 2
Methodology
2.1 Preprocessing Categorical Features

project_grade_category

```
project_data['project_grade_category'].value_counts()
```

```
Grades PreK-2    2002
Grades 3-5       1729
Grades 6-8        785
Grades 9-12       484
Name: project_grade_category, dtype: int64
```

We need to remove the spaces, replace the '-' with '_' and convert all the letters to small.

```
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace(' ','_')
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace('-','_')
project_data['project_grade_category'] = project_data['project_grade_category'].str.lower()
project_data['project_grade_category'].value_counts()
```

```
grades_prek_2    2002
grades_3_5       1729
grades_6_8        785
grades_9_12       484
Name: project_grade_category, dtype: int64
```

Project_subject_categories

```
project_data['project_subject_categories'].value_counts()
```

```
Literacy & Language                         1067
Math & Science                               795
Literacy & Language, Math & Science          679
Health & Sports                              509
```

We need to remove spaces, “the”,
replace ‘&’ with ‘_’
replace ‘,’ with ‘_’

```
project_data['project_subject_categories'] = project_data['project_subject_categories'].str.replace(' The ','')
project_data['project_subject_categories'] = project_data['project_subject_categories'].str.replace(' ','')
project_data['project_subject_categories'] = project_data['project_subject_categories'].str.replace('&','_')
project_data['project_subject_categories'] = project_data['project_subject_categories'].str.replace(',','_')
project_data['project_subject_categories'] = project_data['project_subject_categories'].str.lower()
project_data['project_subject_categories'].value_counts()
```

```
literacy_language                  1067
math_science                        795
literacy_language_math_science      679
health_sports                       509
```

Teacher_prefix

```
project_data['teacher_prefix'].value_counts()
```

```
Mrs.       2560
Ms.        1845
Mr.         495
Teacher     100
Name: teacher_prefix, dtype: int64
```

Remove '.' and convert all characters to small

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.replace('.','')
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.lower()
project_data['teacher_prefix'].value_counts()
```

```
mrs        2560
ms         1845
mr          495
teacher     100
Name: teacher_prefix, dtype: int64
```

Project_subject_subcategories

```
project_data['project_subject_subcategories'].value_counts()
```

```
Literacy                                  449
Literacy, Mathematics                     368
Literature & Writing, Mathematics         293
Literacy, Literature & Writing            234
```

Same process as mentioned in project_subject_categories

```
project_data['project_subject_subcategories'] = project_data['project_subject_subcategories'].str.replace(' The ','')
project_data['project_subject_subcategories'] = project_data['project_subject_subcategories'].str.replace(' ','')
project_data['project_subject_subcategories'] = project_data['project_subject_subcategories'].str.replace('&','_')
project_data['project_subject_subcategories'] = project_data['project_subject_subcategories'].str.replace(',','_')
project_data['project_subject_subcategories'] = project_data['project_subject_subcategories'].str.lower()
project_data['project_subject_subcategories'].value_counts()
```

```
literacy                                449
literacy_mathematics                    368
literature_writing_mathematics          293
literacy_literature_writing             234
```

School_state

```
project_data['school_state'].value_counts()
```

```
CA    707
TX    352
NY    342
```

Convert all the characters to small

```
project_data['school_state'] = project_data['school_state'].str.lower()
project_data['school_state'].value_counts()
```

```
ca    707
tx    352
ny    342
```

Essay

```
print(9, project_data['essay'].values[9])
```

```
printing some random essay
9 Over 95% of my students are on free or reduced lunch.  I have a few who are homeless, but despite that, they come to school
with an eagerness to learn.  My students are inquisitive eager learners who  embrace the challenge of not having great books
and other resources  every day.  Many of them are not afforded the opportunity to engage with these big colorful pages of a b
ook on a regular basis at home and they don't travel to the public library.  \r\nIt is my duty as a teacher to do all I can t
```

As the feature ‘essay’ is a text feature, there are few preprocessing functions to be done such as replacing phrases, removing stop words, removing special characters including numbers, shapes etc.

Removing and replacing specific and general phrases

```
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

Removing stopwords

```
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

Removing special characters

```
from tqdm import tqdm
def preprocess_text(text_data):
    preprocessed_text = []
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_text.append(sent.lower().strip())
    return preprocessed_text
```

```
preprocessed_essays = preprocess_text(project_data['essay'].values)
```

```
100%|
```

After preprocessing

```
print(9, preprocessed_essays[9])
```

```
printing some random reviews
9 95 students free reduced lunch homeless despite come school eagerness learn students inquisitive eager learners embrace cha
llenge not great books resources every day many not afforded opportunity engage big colorful pages book regular basis home no
t travel public library duty teacher provide student opportunity succeed every aspect life reading fundamental students read
books boosting comprehension skills books used read alouds partner reading independent reading engage reading build love read
ing reading pure enjoyment introduced new authors well old favorites want students ready 21st century know pleasure holding g
```

2.2 Preprocessing numerical features

Price

```
project_data['price'].head()
```

```
0    154.60
1    299.00
2    516.85
3    232.90
4     67.98
Name: price, dtype: float64
```

Applying standard scalar, it standardizes the feature by removing mean and scales to unit variance.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(project_data['price'].values.reshape(-1, 1))
project_data['std_price']=scaler.transform(project_data['price'].values.reshape(-1, 1) )
```

```
project_data['std_price'].head()
```

```
0   -0.393708
1   -0.010053
2    0.568751
3   -0.185673
4   -0.623847
Name: std_price, dtype: float64
```

Applying minmax scalar, it rescales the data set such that all feature values are in the range [0, 1]

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(project_data['price'].values.reshape(-1, 1))
project_data['nrm_price']=scaler.transform(project_data['price'].values.reshape(-1, 1))
```

```
project_data['nrm_price'].head()
```

```
0    0.015320
1    0.029763
2    0.051554
3    0.023152
4    0.006656
Name: nrm_price, dtype: float64
```

2.3 Data Vectorization

Vectorization of the data features helps in converting the algorithm from operating on individual elements one at a time, to operating on a batch of values in a single operation. Which makes our model to execute fast and accurate.

Before vectorization, data is split to Train data, CV data and Test data using traintest split. 50000 data points are considered.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(22445, 8) (22445,)
(11055, 8) (11055,)
(16500, 8) (16500,)
```

After splitting the data, Vectorization of data is done, encoding the text feature using BOW, and encoding the remaining features categorical and numerical features using count vectorizer and normalizer.

Encoding the text feature ‘essay’ using Bag of Words

```
vectorizer_essay_bow = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer_essay_bow.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer_essay_bow.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_essay_bow.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_essay_bow.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
After vectorizations
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
```

Encoding the categorical features using count vectorizer

```
vectorizer_clean_sub = CountVectorizer()
vectorizer_clean_sub.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories = vectorizer_clean_sub.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcategories = vectorizer_clean_sub.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories = vectorizer_clean_sub.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_subcategories.shape, y_train.shape)
print(X_cv_clean_subcategories.shape, y_cv.shape)
print(X_test_clean_subcategories.shape, y_test.shape)
print(vectorizer.get_feature_names())
```

```
After vectorizations
(22445, 28) (22445,)
(11055, 28) (11055,)
(16500, 28) (16500,)
['appliedsciences', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice', 'earlydevelopment', 'e
conomics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness', 'health_li
fescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'mathematics', 'music', 'nutritioneducati
on', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts']
```

Remaining categorical features are also encoded using similar way. By just replacing the feature name in the above snippet.

Also for the numerical features by replacing their feature names, but by using normalizer.

Encoding the numerical features using normalizer

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(-1,1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
```

After feature vectorization, all the features are concatenated according to the train, cv and test data. So that hyper parameter tuning is done the concatenated features.

2.4 Hyperparamter tuning

Hyper parameter tuning is done using grid search cv, few default parameters are considered. Three different classification techniques are used to find out the best AUC score.

1. Naïve bayes

   Parameters considered for naïve bayes

```
train_auc = []
cv_auc = []
log_alpha = []
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000]
for i in tqdm(alpha):
    neigh = MultinomialNB(alpha=i, class_prior=[0.5,0.5])
    neigh.fit(X_tr, y_train)
```

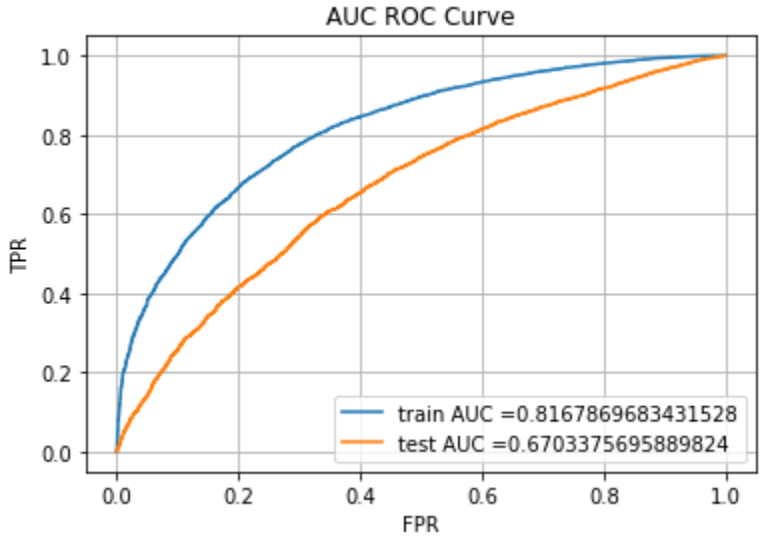
   Using grid search, the best alpha is at 0.1, based on this parameter, AUC curve is plotted and score is known

```
neigh = MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=None)
neigh.fit(X_tr, y_train)

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC ROC Curve")
plt.grid()
plt.show()
```



2. Decision Trees

   Parameters considered for DT are max depth and min samples split

```
DT = DecisionTreeClassifier(class_weight='balanced')
parameters = {'max_depth':[1,5,10,50],'min_samples_split':[5,10,100,500]}
clf = GridSearchCV(DT, parameters, cv=3, scoring='roc_auc')
clf.fit(X_tr_tfidf, y_train)
```

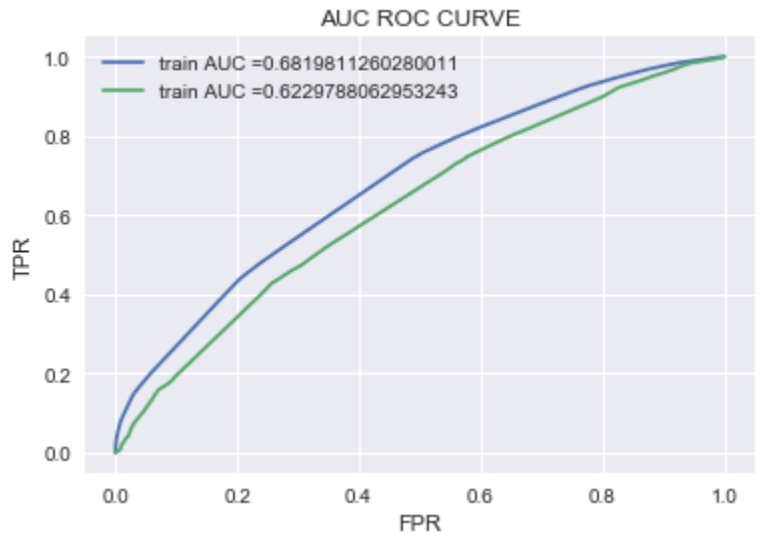
   Using grid search, the best parameters are max depth is at 10 and min samples split is at 500

```
dt = DecisionTreeClassifier(max_depth = 10, min_samples_split = 500, class_weight = 'balanced')
dt.fit(X_tr_tfidf, y_train)

y_train_pred = batch_predict(dt, X_tr_tfidf)
y_test_pred = batch_predict(dt, X_te_tfidf)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC ROC CURVE")
plt.grid(True)
plt.show()
```



3. Gradient Boosting Decision Trees

   Parameters considered for GBDT are learning rate and n estimators

```
GBDT = LGBMClassifier(boosting_type='gbdt',class_weight='balanced')
parameters = {'learning_rate':[0.01,0.1,0.2,0.3],'n_estimators':[5,10,100,500]}
clf = GridSearchCV(GBDT, parameters, cv=3, scoring='roc_auc',n_jobs=-1)
clf.fit(X_tr_tfidf, y_train)
```

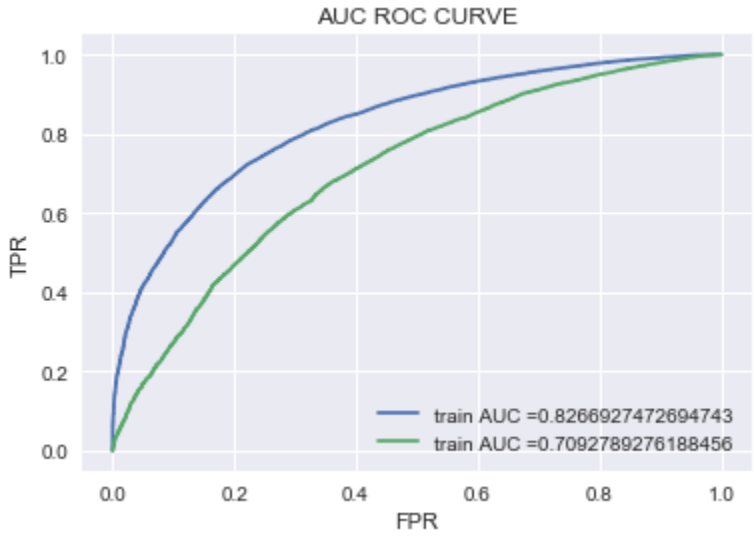
   Using grid search, the best parameters are learning rate at 0.01 and n estimators at 500.

```
gbdt = LGBMClassifier(learning_rate = 0.01, n_estimators = 500, class_weight = 'balanced')
gbdt.fit(X_tr_tfidf, y_train)

y_train_pred = batch_predict(gbdt, X_tr_tfidf)
y_test_pred = batch_predict(gbdt, X_te_tfidf)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("AUC ROC CURVE")
plt.grid(True)
plt.show()
```



For GBDT, instead of Xgboost classifier, Light gbm classifier is used as it takes less time to execute the model than Xgboost classifier.

After plotting the AUC curves, we get the train and the test score, based on those scores, we can compare our models and find out the best classification technique which fits our model.

Chapter 3:

Conclusion

In the given context of our problem statement, we want to find out the classification model which gives us best AUC score. It measures how true positive rate and false positive rate trade off, AUC is considered as a best metric as the estimate of the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance.

model	Train AUC	Test AUC
Naive Bayes	0.816	0.670
Decision Trees	0.681	0.622
Gradient Boosting DT	0.826	0.790

From the above comparison, we can consider that Gradient Boosting Decision Trees using Light gbm classifier is better than Naïve Bayes and Decision Trees.