HW5_Solution

November 29, 2021

[1]:
```
import os
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
```

0.1 Changing HTML Text to Plain Text

[2]:
```
#Read our data file
df_train = pd.read_csv('stack_stats_2020_train.csv')
df_test = pd.read_csv('stack_stats_2020_test.csv')
```

[3]:
```
#Change HTML Text to Plain text using get_text() function from BeautifulSoup
#Manually cleaned up newline tag \n and tab tag \t.

df_train['Body'] = df_train['Body'].apply(lambda text: BeautifulSoup(text,'html.
 ↪parser').get_text())
df_train['Body'] = df_train['Body'].apply(lambda text: text.replace('\n',''))
df_train['Body'] = df_train['Body'].apply(lambda text: text.replace('\t',''))

#Cleaning Tags

df_train['Tags'] = df_train['Tags'].apply(lambda text: text.replace('>',' '))
df_train['Tags'] = df_train['Tags'].apply(lambda text: text.replace('<',''))

#Repeat the same process for test dataset

df_test['Body'] = df_test['Body'].apply(lambda text: BeautifulSoup(text,'html.
 ↪parser').get_text())
df_test['Body'] = df_test['Body'].apply(lambda text: text.replace('\n',''))
df_test['Body'] = df_test['Body'].apply(lambda text: text.replace('\t',''))

#Cleaning Tags

df_test['Tags'] = df_test['Tags'].apply(lambda text: text.replace('>',' '))
df_test['Tags'] = df_test['Tags'].apply(lambda text: text.replace('<',''))
```

[4]:
```
df_train.head(5)
```

[4]:
```
        Id  Score                                               Body  \
0   495560      1  I have a set of data that I am transforming us…
1   489896      0  We are sending a one bit message to someone.  …
2   497951      2  I am aware that there is a similar post: Vecto…
3   478542      2  I have a Poisson distributed glm where I have …
4   458388      0  1) how do i decide which transformation or sca…

                                                Title  \
0     R: emmeans back tranform clr data using clrInv
1  Trying to determine the failure rate of redund…
2  How to derive categorical cross entropy update…
3  Learning more about glm parameters, how to dig…
4  Is there I guide to decide which transformatio…

                                                 Tags
0                      r mixed-model linear lsmeans
1                                probability python
2                             logistic cross-entropy
3           generalized-linear-model interpretation
4  python data-transformation dataset feature-eng…
```

0.2 Basic Text Cleaning and Merging into a single Text data

0.2.1 Change to Lower Case, Remove puncuation, digits,

[5]:
```
#Change to Lowercase

df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].
↪applymap(str.lower)
df_test[['Body','Title','Tags']] = df_test[['Body','Title','Tags']].
↪applymap(str.lower)
```

[6]:
```
#Remove Punctations

from string import punctuation

def remove_punctuation(document):

    no_punct = ''.join([character for character in document if character not in␣
↪punctuation])

    return no_punct
```

[7]:
```
df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].
↪applymap(remove_punctuation)
```

```
df_test[['Body','Title','Tags']] = df_test[['Body','Title','Tags']].
↪applymap(remove_punctuation)
```

[8]:
```
df_train_temp = df_train.copy()
```

[9]:
```
#Remove Digits

def remove_digit(document):

    no_digit = ''.join([character for character in document if not character.
↪isdigit()])

    return no_digit
```

[10]:
```
df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].
↪applymap(remove_digit)
df_test[['Body','Title','Tags']] = df_test[['Body','Title','Tags']].
↪applymap(remove_digit)
```

0.2.2 Tokenization and Remove Stopwords and do stemming

[11]:
```
from nltk.tokenize import word_tokenize
import nltk

df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].
↪applymap(word_tokenize)
df_test[['Body','Title','Tags']] = df_test[['Body','Title','Tags']].
↪applymap(word_tokenize)
```

[12]:
```
#Remove Stopwords

from nltk.corpus import stopwords

stop_words = set(stopwords.words('english'))

def remove_stopwords(document):

    words = [word for word in document if not word in stop_words]

    return words
```

[13]:
```
df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].
↪applymap(remove_stopwords)
df_test[['Body','Title','Tags']] = df_test[['Body','Title','Tags']].
↪applymap(remove_stopwords)
```

[14]:
```
#We use porter stemming

from nltk.stem import PorterStemmer

porter = PorterStemmer()

def stemmer(document):

    stemmed_document = [porter.stem(word) for word in document]

    return stemmed_document
```

[15]:
```
df_train[['Body','Title','Tags']] = df_train[['Body','Title','Tags']].
 ↪applymap(stemmer)
df_test[['Body','Title','Tags']] = df_test[['Body','Title','Tags']].
 ↪applymap(stemmer)
```

0.2.3 Treat Three text data independently and merge into one column

[16]:
```
#Treat Three types of data independently

def add_body(document):

    added_document = [word + '_body' for word in document]

    return added_document

def add_title(document):

    added_document = [word + '_title' for word in document]

    return added_document

def add_tags(document):

    added_document = [word + '_tags' for word in document]

    return added_document
```

[17]:
```
df_train['Body'] = df_train['Body'].apply(add_body)
df_train['Title'] = df_train['Title'].apply(add_title)
df_train['Tags'] = df_train['Tags'].apply(add_tags)

df_test['Body'] = df_test['Body'].apply(add_body)
df_test['Title'] = df_test['Title'].apply(add_title)
df_test['Tags'] = df_test['Tags'].apply(add_tags)
```

[18]:
```
cols = ['Body', 'Title', 'Tags']

#Create a column named 'text'
df_train['text'] = df_train['Body'] + df_train['Title'] + df_train['Tags']
df_test['text'] = df_test['Body'] + df_test['Title'] + df_test['Tags']

#df_train.drop(columns = cols, inplace = True)
#df_test.drop(columns = cols, inplace = True)
```

0.2.4 Detokenize and convert to document term matrices

[19]:
```
#Merge Three text column into one column and detokenize

from nltk.tokenize.treebank import TreebankWordDetokenizer
from sklearn.feature_extraction.text import CountVectorizer

text_train = df_train['text'].apply(TreebankWordDetokenizer().detokenize)
countvec_train = CountVectorizer(min_df = 0.015)
sparse_dtm_train = countvec_train.fit_transform(text_train)
```

[20]:
```
text_test = df_test['text'].apply(TreebankWordDetokenizer().detokenize)
sparse_dtm_test = countvec_train.transform(text_test)
```

[21]:
```
dtm_train = pd.DataFrame(sparse_dtm_train.toarray(), columns=countvec_train.
 ↪get_feature_names(),
                         index=df_train.index)
dtm_test =  pd.DataFrame(sparse_dtm_test.toarray(), columns=countvec_train.
 ↪get_feature_names(),
                         index=df_test.index)
```

[22]:
```
dtm_train.sum().sort_values(ascending=False)
```

[22]:
```
l_body                    27086
use_body                  20319
model_body                20112
data_body                 17301
variabl_body              12697
                          …
maximumlikelihood_tags      302
bayesian_title              301
surviv_tags                 298
classif_title               296
pca_tags                    295
Length: 764, dtype: int64
```

[23]:
```
dtm_test.sum().sort_values(ascending=False)
```

[23]:
```
l_body           9792
use_body         8519
model_body       8134
data_body        7358
variabl_body     5312
                 …
train_title       124
somehow_body      123
classif_title     119
valid_title       115
problem_title     113
Length: 764, dtype: int64
```

0.2.5 Change dependent variable to binary variable

[24]:
```
y_train = (df_train['Score']>=1).astype(int)
y_test = (df_test['Score']>=1).astype(int)
```

[25]:
```
y_train.value_counts()
```

[25]:
```
0    9684
1    9563
Name: Score, dtype: int64
```

[26]:
```
df_train['GoodQuestion'] = y_train
df_test['GoodQuestion'] = y_test
df_train.drop(columns = ['Score','Id'], inplace = True)
df_test.drop(columns = ['Score','Id'], inplace = True)
```

0.3 Answer for (a) (30 points)

Since this is a open-ended question, data cleaning process depends on who is doing the process. However, there are some necessary things to be done. Following are the lists of such necessary process.

List A(Basic Cleaning Process)

- Turn to lower case
- Remove punctations and digits
- stem document
- Remove stopwords
- Remove infrequent words by using countVectorizer.

List B(HTML Cleaning Process)

- Remove HTML
- Remove remaining space charaters in html

List C(Data Related Process)

- Change our dependent variable to binary variable.(i.e. If score is greater than equal to 1 than we change to 1)
- Treat column 'Body', 'Title', 'Tags' independently. You should also merge those columns to one column.

0.3.1 Grading Rubrics

- (-3) For not carrying out the steps in List A.
- (-4) For not carrying out the steps in List B and C.
- (-3) You could either merge the training and test data set or just use training set to fit countVectorizer function. In the latter, you would gain full credit if you obtain dtm of the test set by using countVectorizer.transform(). For the former, you would gan full credit if you split the merged data set back to training and test data set as it were given. Otherwise, you would lose 3 points. For example, if you split the merged dtm by using randomsplit function, you would be deducted.

0.3.2 (b)

1) Logistic Regression

[27]:
```
import statsmodels.api as sm

logreg = sm.Logit(y_train,dtm_train).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.647801
         Iterations 8
```

Decision Tree Classifier

[28]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

grid_values = {'ccp_alpha': np.linspace(0, 0.01, 10)}

dtc = DecisionTreeClassifier(random_state=88)
dtc_cv = GridSearchCV(dtc, param_grid=grid_values, cv=10).fit(dtm_train,␣
↪y_train)
```

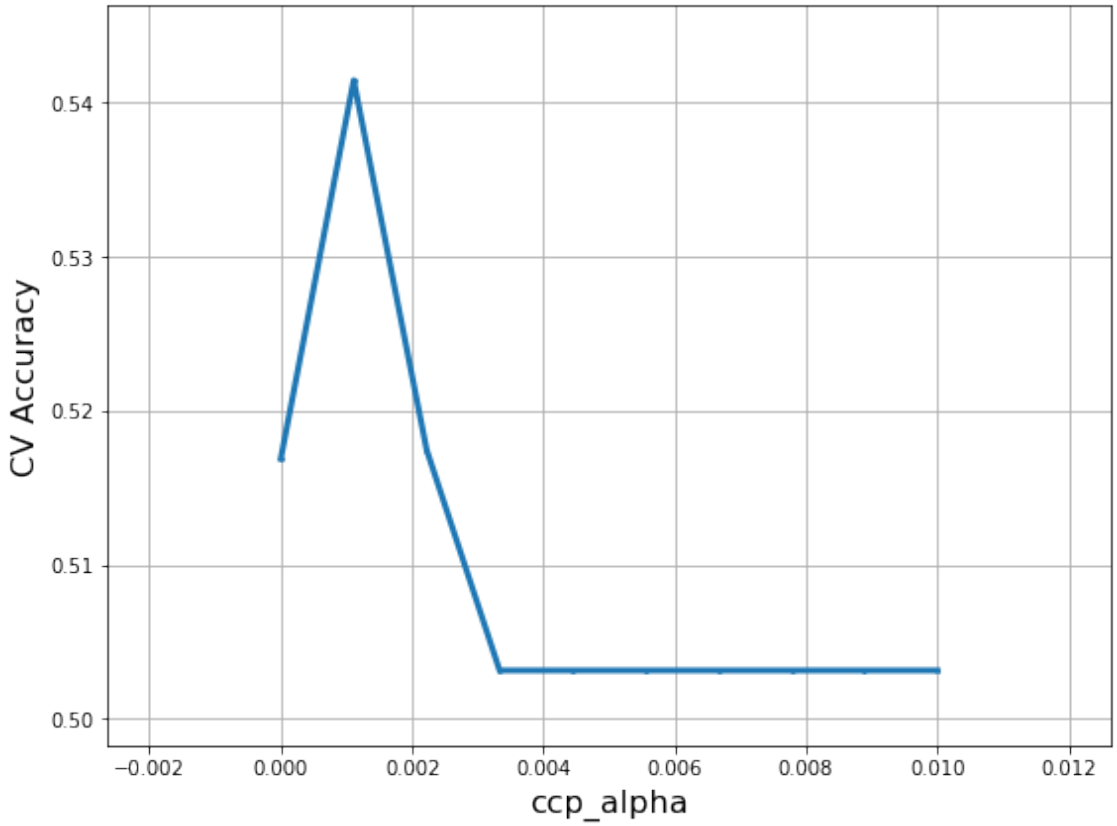
[29]:
```
ccp_alpha = dtc_cv.cv_results_['param_ccp_alpha'].data
ACC_scores = dtc_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('ccp_alpha', fontsize=16)
plt.ylabel('CV Accuracy', fontsize=16)
plt.scatter(ccp_alpha, ACC_scores, s=3)
plt.plot(ccp_alpha, ACC_scores, linewidth=3)
plt.grid(True, which='both')

plt.tight_layout()
```

```
plt.show()

print('Best ccp_alpha', dtc_cv.best_params_)
```




```
Best ccp_alpha {'ccp_alpha': 0.0011111111111111111}
```

LDA

[30]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis()
lda.fit(dtm_train, y_train)
```

[30]:
```
LinearDiscriminantAnalysis()
```

Random Forest Classifier

[31]:
```
from sklearn.ensemble import RandomForestClassifier

import time

grid_values = {'max_features': np.linspace(20,40,5, dtype='int32'),
```

```
                'min_samples_leaf': [5],
                'n_estimators': [300],
                'random_state': [88]}

tic = time.time()

rf = RandomForestClassifier()
rf_cv = GridSearchCV(rf, param_grid=grid_values, cv=5)
rf_cv.fit(dtm_train, y_train)

toc = time.time()

print('time:', round(toc-tic, 2),'s')
```

```
time: 1470.58 s
```

Comparison on the test set

[32]:
```
#Baseline Model
from sklearn.metrics import confusion_matrix


default_false = np.sum(y_train==0)
default_true = np.sum(y_train==1)
print(pd.Series({'0': default_false, '1': default_true}))
```

```
0    9684
1    9563
dtype: int64
```

[33]:
```
#Baseline Model Statistics
baseline_acc = default_false/(default_true+default_false)
baseline_TPR = 0
baseline_FPR = 0
baseline_PRE = 0
```

[34]:
```
#Logistic Regression Model Statistics
log_prob = logreg.predict(dtm_test)
log_pred = pd.Series([1 if x > 0.5 else 0 for x in log_prob], index=log_prob.
 ↪index)

cm = confusion_matrix(y_test, log_pred)
print ("Confusion Matrix : \n", cm)

log_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
log_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
log_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
log_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
```

```
Confusion Matrix :
 [[2435 1791]
 [1859 2164]]
```

[35]:
```
#DTC Statistics

dtc_pred = dtc_cv.best_estimator_.predict(dtm_test)

cm = confusion_matrix(y_test, dtc_pred)
print ("Confusion Matrix : \n", cm)

dtc_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
dtc_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
dtc_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
dtc_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
```

```
Confusion Matrix :
 [[3243  983]
 [2789 1234]]
```

[36]:
```
#LDA Statistics

y_pred = lda.predict(dtm_test)
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix: \n", cm)

lda_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
lda_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
lda_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
lda_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
```

```
Confusion Matrix:
 [[2617 1609]
 [2006 2017]]
```

[37]:
```
#RFC Statistics

y_pred = rf_cv.best_estimator_.predict(dtm_test)
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix: \n", cm)

rf_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
rf_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
rf_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
rf_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
```

```
Confusion Matrix:
 [[2583 1643]
```

```
[1828 2195]]
```

[38]:
```
#Create Comparison Table

comparison_data = {'Baseline':[baseline_acc,baseline_TPR,baseline_FPR,␣
↪baseline_PRE],
                   'Logistic Regression':[log_acc,log_TPR,log_FPR, log_PRE],
                   'Decision Tree Classifier':[dtc_acc,dtc_TPR,dtc_FPR,dtc_PRE],
                   'Random Forest with CV':[rf_acc,rf_TPR, rf_FPR,rf_PRE],
                  'Linear Discriminant Analysis':[lda_acc,lda_TPR,␣
↪lda_FPR,lda_PRE]}

comparison_table = pd.DataFrame(data=comparison_data, index=['Accuracy', 'TPR',␣
↪'FPR','PRE']).transpose()
comparison_table.style.set_properties(**{'font-size': '12pt',}).
↪set_table_styles([{'selector': 'th', 'props': [('font-size', '10pt')]}])
comparison_table
```

[38]:

	Accuracy	TPR	FPR	PRE
Baseline	0.503143	0.000000	0.000000	0.000000
Logistic Regression	0.557522	0.537907	0.423805	0.547155
Decision Tree Classifier	0.542732	0.306736	0.232608	0.556608
Random Forest with CV	0.579222	0.545613	0.388784	0.571912
Linear Discriminant Analysis	0.561765	0.501367	0.380738	0.556260

0.4 Bootstrap

[39]:
```
#Implement Bootstrap

#Define bootstrap function
def bootstrap_validation_logreg(test_data, test_label, model, sample=500,␣
↪random_state=66):
    tic = time.time()
    n_sample = sample
    output_array=np.zeros([n_sample, 4])
    output_array[:]=np.nan
    print(output_array.shape)
    for bs_iter in range(n_sample):
        bs_index = np.random.choice(test_data.index, len(test_data.index),␣
↪replace=True)
        bs_data = test_data.loc[bs_index]
        bs_label = test_label.loc[bs_index]
        bs_prob = model.predict(bs_data)
        bs_pred = pd.Series([1 if x > 0.5 else 0 for x in bs_prob],␣
↪index=bs_prob.index)

        cm = confusion_matrix(test_label, bs_pred)
```

```
        log_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
        log_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
        log_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
        log_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
        output_array[bs_iter,:]=np.array([log_acc,log_TPR,log_FPR,log_PRE])

    output_df = pd.DataFrame(output_array)
    return output_df
```

[40]:
```
#Let's Visualize our result
# CI of ACC_boosting - log_acc, TPR_boosting - log_TPR, FPR_boosting - log_FPR
bs_output = bootstrap_validation_logreg(dtm_test, y_test, logreg,sample = 1000)
```

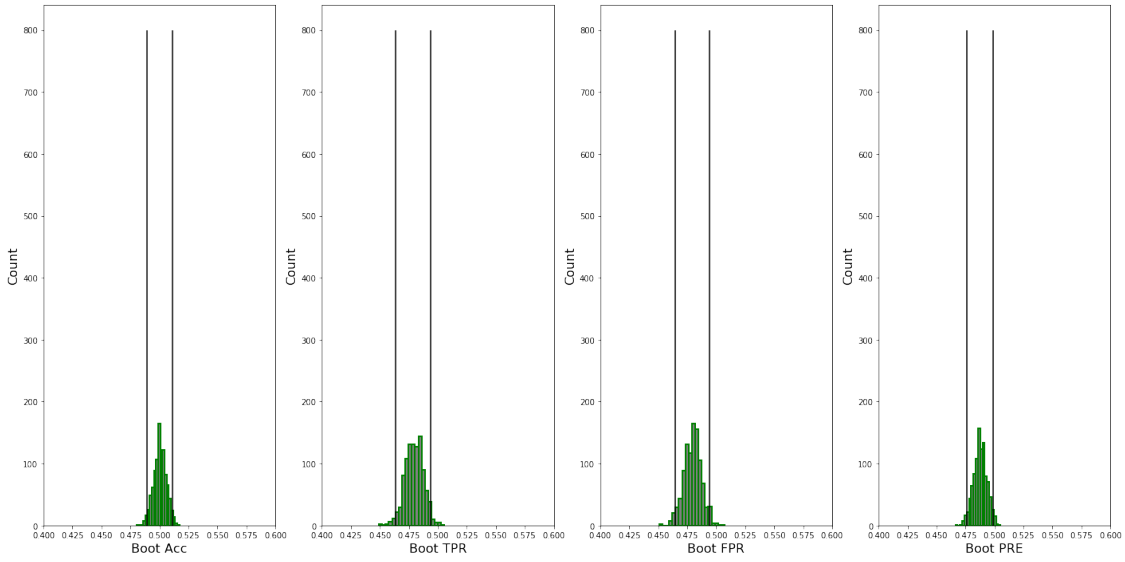
```
(1000, 4)
```

[41]:
```
CI_acc = np.quantile(bs_output.iloc[:,0],np.array([0.025,0.975]))
CI_TPR = np.quantile(bs_output.iloc[:,1],np.array([0.025,0.975]))
CI_FPR = np.quantile(bs_output.iloc[:,2],np.array([0.025,0.975]))
CI_PRE = np.quantile(bs_output.iloc[:,3],np.array([0.025,0.975]))

mean_acc = np.mean(bs_output.iloc[:,0])
std_acc = np.std(bs_output.iloc[:,0])
mean_TPR = np.mean(bs_output.iloc[:,1])
std_TPR = np.std(bs_output.iloc[:,1])
mean_FPR = np.mean(bs_output.iloc[:,2])
std_FPR = np.std(bs_output.iloc[:,2])
mean_PRE = np.mean(bs_output.iloc[:,3])
std_PRE = np.std(bs_output.iloc[:,3])


fig, axs = plt.subplots(ncols=4, figsize=(24,12))
#Plot Accuracy
axs[0].set_xlabel('Boot Acc', fontsize=16)
axs[0].set_ylabel('Count', fontsize=16)
axs[0].hist(bs_output.iloc[:,0], bins=20,edgecolor='green', linewidth=2,color =␣
 ↪"grey")
axs[0].set_xlim([0.4,0.6])
axs[0].vlines(x=CI_acc[0], ymin = 0, ymax =800, color = "black")
axs[0].vlines(x=CI_acc[1], ymin = 0, ymax =800, color = "black")
#Plot TPR
axs[1].set_xlabel('Boot TPR', fontsize=16)
axs[1].set_ylabel('Count', fontsize=16)
axs[1].hist(bs_output.iloc[:,1], bins=20,edgecolor='green', linewidth=2,color =␣
 ↪"grey")
axs[1].set_xlim([0.4,0.6])
axs[1].vlines(x=CI_TPR[0], ymin = 0, ymax =800, color = "black")
axs[1].vlines(x=CI_TPR[1], ymin = 0, ymax =800, color = "black")
```

```
#Plot FPR
axs[2].set_xlabel('Boot FPR', fontsize=16)
axs[2].set_ylabel('Count', fontsize=16)
axs[2].hist(bs_output.iloc[:,2], bins=20,edgecolor='green', linewidth=2,color =␣
 ↪"grey")
axs[2].set_xlim([0.4,0.6])
axs[2].vlines(x=CI_FPR[0], ymin = 0, ymax =800, color = "black")
axs[2].vlines(x=CI_FPR[1], ymin = 0, ymax =800, color = "black")
#Plot Precision
axs[3].set_xlabel('Boot PRE', fontsize=16)
axs[3].set_ylabel('Count', fontsize=16)
axs[3].hist(bs_output.iloc[:,3], bins=20,edgecolor='green', linewidth=2,color =␣
 ↪"grey")
axs[3].set_xlim([0.4,0.6])
axs[3].vlines(x=CI_PRE[0], ymin = 0, ymax =800, color = "black")
axs[3].vlines(x=CI_PRE[1], ymin = 0, ymax =800, color = "black")
plt.show()
```



[42]:
```
# Let's make a table for this
bootstrap_data = {'Accuracy':[CI_acc[0],CI_acc[1],mean_acc,std_acc],'TPR':
 ↪[CI_TPR[0],CI_TPR[1],mean_TPR,std_TPR],
                  'FPR':[CI_FPR[0],CI_FPR[1],mean_FPR,std_FPR],'PRE':
 ↪[CI_PRE[0],CI_PRE[1],mean_PRE,std_PRE]}
bootstrap_table = pd.DataFrame(data = bootstrap_data, index = ['0.025␣
 ↪quantile','0.975 quantile',
                                                                'Mean','Standard␣
 ↪Deviation']).transpose()
```

```
bootstrap_table.style.set_properties(**{'font-size': '12pt',}).
 ↪set_table_styles([{'selector': 'th', 'props': [('font-size', '10pt')]}])
bootstrap_table
```

[42]:

	0.025 quantile	0.975 quantile	Mean	Standard Deviation
Accuracy	0.489393	0.510974	0.500458	0.005450
TPR	0.463827	0.493910	0.479643	0.007785
FPR	0.464500	0.494321	0.479726	0.007457
PRE	0.476183	0.498621	0.487651	0.005688

0.4.1 Answer for (b) (40 points)

You can select any 4 models that you want. You should explain the details of your training process and report final comparisons on the test data. Otherwise you will be deducted.

Here, we selected logistic regression, decision tree classifier, LDA and Random forest. In logistic regression and LDA, we set our good question binary variable as our dependent variable and our processed text data as our independent variable. In decision tree classifier, we implemented grid search over parameter ccp_alpha. We implemented 10 fold cross validation with four different parameter values. Also, in random forest model, we implemented grid search over parameter max_features parameter. We implemented 10 fold cross validation with four different parameter values with parameter n_estimator = 300. If you look at the final comparison results above, we can see that logistic regression good performance compared to the other models in the table. It has good accuracy and TPR. So we choose the logistic regression model as our final and implemented bootstrap to this model. We can see that there is significant difference between boosting model and our random forest model.

0.4.2 Grading Rubrics

- (-2) For each improper model. For example, if they used random forest regressor instead of random forest classifier in this classification problem they would be deducted. Another improper models would be linear regression and decision tree regressor. Maximum Total deduction in this criteria is 8 points.
- (-4) If student didn't describe his/her training process or didn't give reasonable explanation for his/her parameter selection, then he/she would be deducted 3 points for each non-explained model. Performing Grid search with cross validation would be a reasonable way to select parameters. Maximum Total deduction in this criteria is 16 points.
- (-6) Lack of clarification of their final comparisons on the test set.
- (-10) Not or improperly implementing bootstrap method to their final model. Students have to make a modification to lab code to do this. If the student didn't present the bootstrap results in a nice form, deduct 5 points.

0.4.3 (c)

0.4.4 Answer for (c) (30 points)

This is an open-ended question. If you give us a reasonable explanation of the grading criteria below, you would gain full credit. Here is our answer for this question.

Explanation of which precise criteria I would use for model selection. First of all, I would select a model with the highest precision score. Since our goal is to maximize the probability that the top question is useful, it is important to make our positive prediction as accurate as possible. Precision actually measures the probability of correct detection of positive values. Now let's take a look at the comparison table which we created on problem (b).

[43]:
```
comparison_table
```

[43]:

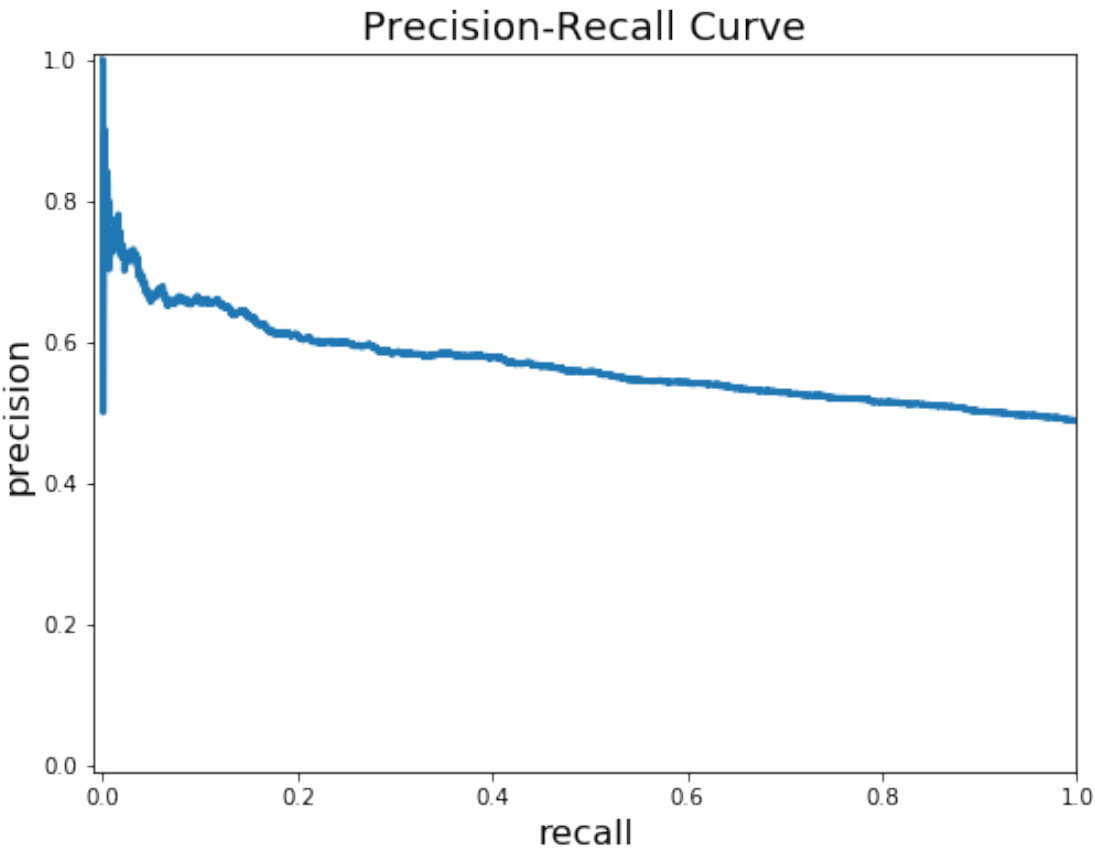
	Accuracy	TPR	FPR	PRE
Baseline	0.503143	0.000000	0.000000	0.000000
Logistic Regression	0.557522	0.537907	0.423805	0.547155
Decision Tree Classifier	0.542732	0.306736	0.232608	0.556608
Random Forest with CV	0.579222	0.545613	0.388784	0.571912
Linear Discriminant Analysis	0.561765	0.501367	0.380738	0.556260

How could we improve our model? We can see that random forest with cv model has the highest precision score. However, since it takes so long to retrain it we use logistic regression model instead. Logistic regression model also has high accuracy and TPR and precision score. We could increase precision by changing the threshold value of our model. According to our test set, about 50% of the questions are considered to be a good question. So if we are given 15 questions, 7.5 questions would be considered as good question on average. Among these 7.5 positive cases, we want at least one positive prediction. Using back-of-the-envelope 'on average' style of analysis, we want our model's TPR >= 1/7.5. Let's draw precision-recall curve to show the relationship between precision and recall(TPR) score. If we take a look at precision-recall curve, you can see that precision increases as recall(TPR) decreases. So we should decrease recall score as much as we can by controlling our threshold parameter. However, notice that we should at least satisfy TPR >= 1/7.5 = 0.133333. So, let's find a threshold value that makes TPR close to 0.15. (We are giving some safe buffer to the TPR.) Observing the below result, threshold = 0.68 would make our logistic regression model better in precision score and actually it did. Our precision has changed from 0.55 to 0.63 and also FPR has been reduced significantly from 0.42 to 0.08.

[44]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc

#Draw Precision recall curve
precision, recall, _ = precision_recall_curve(y_test,log_prob)

plt.figure(figsize=(8, 6))
plt.title('Precision-Recall Curve', fontsize=18)
plt.xlabel('recall', fontsize=16)
plt.ylabel('precision', fontsize=16)
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.plot(recall, precision, lw=3)
plt.show()
```



[45]:
```
temp_recall = recall[1:]>= 0.15
np.nonzero(temp_recall)
```

[45]:
```
(array([   0,    1,    2, …, 7294, 7295, 7296], dtype=int64),)
```

[46]:
```
p = _[7301]
print(p)
```

```
0.6824052051647954
```

[47]:
```
log_pred = pd.Series([1 if x > p else 0 for x in log_prob], index=log_prob.
 ↪index)

cm = confusion_matrix(y_test, log_pred)
print ("Confusion Matrix : \n", cm)

log_acc = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
log_TPR = cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])
log_FPR = cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])
log_PRE = cm.ravel()[3]/(cm.ravel()[1]+cm.ravel()[3])
```

```
print('log_acc:',log_acc)
print('log_TPR:',log_TPR)
print('log_FPR:',log_FPR)
print('log_PRE:',log_PRE)
```

```
Confusion Matrix :
 [[3880  346]
 [3423  600]]
log_acc: 0.5430961328645897
log_TPR: 0.14914243102162567
log_FPR: 0.08187411263606247
log_PRE: 0.6342494714587738
```

0.4.5 Grading Rubrics

- (-10) Not giving reasonable explanation of evaluation metrics that the students used to choose their model. TNR(=1-FPR), FPR, precision might be the possible right answer. These three metrics are actually related to evaluate the model's performance of maximizing the probability that the top question to be useful. If student uses other metrics such as accuracy, they would be deducted by points. Also, from our problem settings, we need our model's TPR to be greater than equal to 1/7.5. If student didn't explain about why this constraint should be satisfied, we will deduct by 5 points.
- (-5) After choosing the model, if students does not utilize the problem structure (15 problems presented on one page), then they would be deducted by 5 points. They should either retrain their model or give us a reasonable explanation of not retraining. In our solution, we changed the threshold value according to our problem structre.
- (-5) Did not compared your model's performance with stack exchange's current approach.
- (-10) If student is justifying him/herself without using numerical results, the student would be deducted by 10 points. This also includes not giving a precise numerical estimate of the increase in the probability that the top question is useful.

[ ]: