Final Project Team Code – Seasonal Sales Analysis
Nomuka Luehr & Shirley Gao

In [1]:

```
import numpy as np
import pandas as pd
from sklearn import datasets

from sklearn.linear_model import LogisticRegression, LinearRegression, RidgeClas
sifier, Ridge
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
%matplotlib inline
```

Data Preparation

In [2]:

```
# read files
orig = pd.read_csv('BlackFriday.csv')
orig.head()
```

Out[2]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years
0	1000001	P00069042	F	170-	10	A	2
0-							
1	1000001	P00248942	F	17	10	A	2
0-							
2	1000001	P00087842	F	17	10	A	2
0-							
3	1000001	P00085442	F	17	10	A	2
4	1000002	P00285442	M	55+	16	C	4+

In [3]:

```
data = orig
```

In [4]:

```
# list out the types of features
data.dtypes
```

Out[4]:

```
User_ID                              int64
Product_ID                          object
```

```
Gender                              object
Age                                 object
Occupation                           int64
City_Category                       object
Stay_In_Current_City_Years          object
Marital_Status                       int64
Product_Category_1                   int64
Product_Category_2                 float64
Product_Category_3                 float64
Purchase                             int64 dtype:
object
```

In [5]:

```
# list out the values for Gender, Age, Stay_In_Current_City_Years, Marital_Statu
s
for col_name in [ 'Gender', 'Age', 'Stay_In_Current_City_Years', 'Marital_Status
']:     print( col_name, ": ", data[col_name].unique()
)
```

```
Gender :  ['F' 'M']
Age :  ['0-17' '55+' '26-35' '46-50' '51-55' '36-45' '18-25']
Stay_In_Current_City_Years :  ['2' '4+' '3' '1' '0']
Marital_Status :  [0 1]
```

In [6]:

```
# convert values for selected features
to_replace = {'Stay_In_Current_City_Years':{'0':0, '1':1, '2':2, '3':3, '4+':4},
              'Marital_Status':{0:'N', 1:'Y'}
          } data =

data.replace(to_replace) In
```

[7]:

```


# dummyise features - Gender, Age, City_Category, Marital_Status for
field in ['Gender','Age','City_Category','Marital_Status']:
# Go through each possible value     for value in data[field].unique():
        # Create a new binary field        data[field + "_" + value] =
pd.Series(data[field] == value, dtype=int)     # Drop the original field
data = data.drop([field], axis=1)

# show new data

data.head() Out[7]:
```

User_ID	Product_ID	Occupation	Stay_In_Current_City_Years	Product_Category_1	Product_

0 1000001 P00069042 10 2 3 **1** 1000001 P00248942 10 2 1 **2** 1000001 P00087842 10 2 12 **3** 1000001 P00085442 10 2 12 **4** 1000002 P00285442 16 4 8

5 rows × 22 columns

In [8]:

```
# data cleaning
# check for missing values
missing_values = data.isnull().sum().sort_values(ascending = False)
missing_values = missing_values[missing_values > 0]/data.shape[0]
print(f'{missing_values *100} %')
```

```
Product_Category_3    69.441029
Product_Category_2    31.062713 dtype:
float64 %
```

In [9]:

```
# copy the data to a new dataframe
# fill the NaN value with 0 value
data0 = data.fillna(0)
```

In [10]:

```
# check for missing values again
# make sure there is no missing value
missing_values = data0.isnull().sum().sort_values(ascending = False)
missing_values = missing_values[missing_values > 0]/data0.shape[0]
print(f'{missing_values *100} %')
```

```
Series([], dtype: float64) %
```

In [11]:

```
# select the features we want to evaluete
feats = ['Age_0-17','Age_18-25','Age_26-35','Age_36-45','Age_46-50','Age_51-55',
'Age_55+','Stay_In_Current_City_Years','Gender_F', 'Gender_M', 'City_Category_A'
,'City_Category_B','City_Category_C','Marital_Status_N','Marital_Status_Y']
# set the labels
# we want to predict the amount of products we should stock for each category in
the future
label1 = ['Product_Category_1'] label2
= ['Product_Category_2'] label3 =
['Product_Category_3']

# split the whole data into 2 data: 90% train data, 10% test data
# we will train our model on train data and evaluate our model with test data

X = data0[feats]
Y = data0[label1 + label2 + label3]
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, random_
state=542) In [12]:
```

```
# use classification to predict whether an instance has a 0 value in Product_Cat
egory_2 & 3
# we skip Product_Category_1 because it does not have 0 value #
train data for field in
['Product_Category_2','Product_Category_3']:      # Go through
each possible value      for value in
Y_train[field].unique():
        # Create a new binary field
        Y_train[field + '_1' ] = pd.Series(Y_train[field] != 0, dtype=int)


# test data for field in
['Product_Category_2','Product_Category_3']:      # Go through
each possible value      for value in Y_test[field].unique():
        # Create a new binary field
        Y_test[field + '_1' ] = pd.Series(Y_test[field] != 0, dtype=int)
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy

/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:16: Set
tingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/panda
s-docs/stable/indexing.html#indexing-view-versus-copy
app.launch_new_instance() In [13]:
```

```
# create new dataframes that drop NaN values for each Product_Category_2 & 3
# we skip Product_Category_1 because it does not contain NaN values
# we will use this for classification
data_PC2 = data.dropna(axis = 0, subset = ['Product_Category_2']) data_PC3
= data.dropna(axis = 0, subset = ['Product_Category_3'])

X_PC2 = data_PC2[feats]
Y_PC2 = data_PC2[label1 + label2 + label3] X_train_PC2, X_test_PC2, Y_train_PC2,
Y_test_PC2 = train_test_split(X_PC2, Y_PC2
, test_size=0.1, random_state=542)

X_PC3 = data_PC3[feats]
Y_PC3 = data_PC3[label1 + label2 + label3] X_train_PC3, X_test_PC3, Y_train_PC3,
Y_test_PC3 = train_test_split(X_PC3, Y_PC3
```

```
, test_size=0.1, random_state=542) In
```

```
[ ]:
```

Baseline

We use the mean value to as the baseline to compare with the performance of our model.

```
In [16]:
```

```
# calculate a baseline
# use mean value to predict
# first we look at Product_Category_1 print("Baseline")
print("")

mae_base = np.sum(np.abs(Y_train[label1] - Y_train[label1].mean()))[0] / len(Y_t
rain[label1]) predict_base = Y_train[label1].mean()[0] *
len(Y_test[label1]) actual_base = np.sum(Y_test[label1])[0]
off_base = (predict_base - actual_base)/actual_base
print("Product_Category_1") print("Mean absolute
error:",round(mae_base, 4))
print("Our predictions are",round(off_base*100, 4),"% off from the actual values
") print("")

mae_base = np.sum(np.abs(Y_train[label2] - Y_train[label2].mean()))[0] / len(Y_t
rain[label2])
predict_base = Y_train[label2].mean()[0] * len(Y_test[label2]) actual_base
= np.sum(Y_test[label2])[0]
off_base = (predict_base - actual_base)/actual_base print("Product_Category_2")
print("Mean absolute error:",round(mae_base, 4))
print("Our predictions are",round(off_base*100, 4),"% off from the actual values
") print("")

mae_base = np.sum(np.abs(Y_train[label3] - Y_train[label3].mean()))[0] / len(Y_t
rain[label3]) predict_base = Y_train[label3].mean()[0] *
len(Y_test[label3]) actual_base = np.sum(Y_test[label3])[0]
off_base = (predict_base - actual_base)/actual_base
print("Product_Category_3") print("Mean absolute
error:",round(mae_base, 4))
print("Our predictions are",round(off_base*100, 4),"% off from the actual values
")
Baseline
```

```
Product_Category_1
Mean absolute error: 2.8898 Our predictions are 0.5645
% off from the actual values
```

```
Product_Category_2
Mean absolute error: 5.5991 Our predictions are -0.5719
% off from the actual values

Product_Category_3
Mean absolute error: 5.3813
Our predictions are 0.3549 % off from the actual values
```

Predict the amount to stock for each prodcut category

Product_Category_1

In [17]:

```
# Forward Feature Selection def check_next_subset( all_feats, labels,
known_good_feats ):

    best_score = 100000     best_subset
= None     for f in all_feats.columns.tolist():

        # If we've selected this feature already, do not consider it again
if f in known_good_feats:                continue

        # Create a copy of the good features, so that we can append the new one
        # This will be the feature subset for this iteration
iter_feat_subset = list(known_good_feats)         iter_feat_subset.append(f)

        # From the dataframe, get the columns of interest         train_subset
= all_feats[iter_feat_subset]

        # complexity control for our model
# control the value fo alpha         a = 0
error = 100000
        alpha_ridge = [1e-15, 1e-10, 1e-8, 1e-4, 1e-3,1e-2, 1, 5, 10, 20]
for i in alpha_ridge:             model = Ridge(alpha=i, normalize=True) # we
select ridge regression as our model
            # we use negative mean absolute error as our score to evaluate the p
erformance of our model
            # first we negate this value, make it a positive number
            # the lower (closer to 0) the better
            # if current score is lower than the best score
            # we set it as the new best score, and set the new alpha value
avg_score = - np.mean( cross_val_score( model, train_subset, labels,
scoring='neg_mean_absolute_error', cv=3 ) )          if avg_score < error:
error = avg_score                a = i

        model = Ridge(alpha=a, normalize=True)
```

```
        avg_score = - np.mean( cross_val_score( model, train_subset, labels, sco
ring='neg_mean_absolute_error', cv=3 ) )

        if avg_score < best_score:
best_score = avg_score              best_subset
= iter_feat_subset      return best_subset,
best_score, a
```

In [18]:

```
# Select the best features to work with
known_good_feats = []
score = 100000

best_score_1 = 10000
best_feats_1 = []
al_1 = 0
for i in range(15): # 15 is the number of all features
    known_good_feats, score, a = check_next_subset( X_train, Y_train[label1], kn
own_good_feats )
    if score < best_score_1:
        best_score_1 = score
        best_feats_1 = known_good_feats
        al_1 = a
    else:
        break
        print("Finished")
print("We select",best_feats_1)
print("Mean absolute error (train data):", round(best_score_1,4))
```

```
We select ['Age_18-25', 'Age_26-35', 'Age_55+', 'Age_0-17', 'Age_36-
45', 'Age_46-50']
Mean absolute error (train data): 2.8745
```

In [19]:

```
# set X to the selected best feats
X_train_best_1 = X_train[best_feats_1]
X_test_best_1 =  X_test[best_feats_1]

# model: ridge regression
model_1 = Ridge(alpha=al_1, normalize=True) model_1.fit(X_train_best_1,
Y_train[label1])

predictions_1 = model_1.predict(X_test_best_1) mae_1 =
metrics.mean_absolute_error( Y_test[label1], predictions_1 )
predict_1 = np.sum(predictions_1) actual_1 =
np.sum(Y_test[label1])[0] off_1 = (predict_1-actual_1)/actual_1
print("Product_Category_1") print("Model: Ridge Regression")
print("Mean absolute error: ", round(mae_1,4)) print("Our predictions
are",round(off_1*100,4),"% off from the actual values")
```

```
Product_Category_1
Model: Ridge Regression
Mean absolute error:  2.8629
Our predictions are 0.5678 % off from the actual values
```

Comparing with the baseline, this model does not significantly improve the performance. Reasons...

Product_Category_2

Ridge Regression

In [20]:

```
# repeat for Product_Category_2
```

In [22]:

```
# look at the relationship between features & Product_Category_2

# Select the best features to work with
known_good_feats = []
score = 100000

best_score_2 = 10000
best_feats_2 = []
al_2 = 0
for i in range(15):
    known_good_feats, score, a = check_next_subset( X_train, Y_train[label2], kn
own_good_feats )
    if score < best_score_2:
        best_score_2 = score
        best_feats_2 = known_good_feats
        al_2 = a
    else:
        break
        print("Finished")
print("We select",best_feats_2)
print("The best mean absolute error we get is", round(best_score_2,4))
```

```
We select ['City_Category_C', 'Age_18-25', 'Age_0-17', 'Age_26-35',
'Age_36-45', 'Marital_Status_N', 'Age_51-55', 'Gender_F']
The best mean absolute error we get is 5.5932
```

In [23]:

```
print("Product_Category_2 - before classification") print("Model:
Ridge Regression")
```

```
# set X to the selected best feats
X_train_best_2 = X_train[best_feats_2]
X_test_best_2 = X_test[best_feats_2]

# model
model_2 = Ridge(alpha=al_2, normalize=True)
model_2.fit(X_train_best_2, Y_train[label2])

predictions_2 = model_2.predict(X_test_best_2)
mae_2 = metrics.mean_absolute_error( Y_test[label2], predictions_2 )
predict_2 = np.sum(predictions_2) actual_2 =
np.sum(Y_test[label2])[0] off_2 = (predict_2-actual_2)/actual_2
```

```
print("Mean absolute error: ", round(mae_2,4)) print("Our predictions
are",round(off_2*100,4),"% off from the actual values")
```

```
Product_Category_2 - before classification
Model: Ridge Regression
Mean absolute error:  5.5874 Our predictions are -0.5811
% off from the actual values
```

Classification In

[24]:

```
### complexity control
```

In [25]:

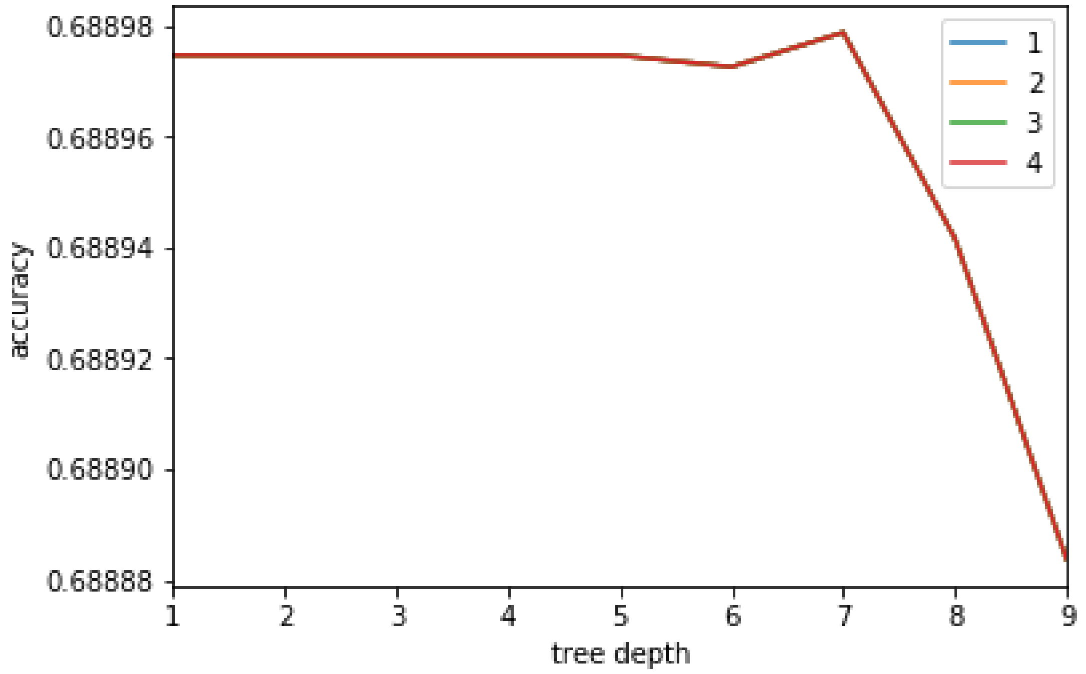
```
# this function return the accuracy from given values of max_depth and min_sampl
es_leaf def cv_eval(X_train, Y_train, d, l):
    # Check above for what the method should do
    model = DecisionTreeClassifier(max_depth=d, min_samples_leaf=l, criterion="e
ntropy")
    model.fit(X_train, Y_train)
    acc = np.mean(cross_val_score(model, X_train, Y_train, cv=3))
return acc  # Return the proper value
```

In [26]:

```
depths_list = range(1, 10)
min_leaf_size_list =  range(1, 5)

df = pd.DataFrame(index=depths_list ,columns=min_leaf_size_list)


for v in min_leaf_size_list:
    l = []
    for x in depths_list:
        l.append(cv_eval(X_train, Y_train['Product_Category_2_1'], x, v))
    df[v] = l

ax = df.plot()
ax.set_xlabel("tree depth")
ax.set_ylabel("accuracy")
```

Out[26]:

```
Text(0, 0.5, 'accuracy')
```



From the graph, we choose max_depth = 7 We do not set min_samples_leaf because it does not make a significant difference.

In [27]:

```
# Get the max depth from the maximum value of accuracy
i =np.argmax(l)
md = depths_list[i]
```

In [ ]:

```

```

In [28]:

```
# split the train data into sub train & test data sets to do classification
X_train_sub, X_test_sub, Y_train_sub, Y_test_sub = train_test_split(
    X_train, Y_train['Product_Category_2_1'], test_size=0.2)
```

In [29]:

```
# Decision Tree Classifier - whether a customer will buy Product_Category_2
dec_tree_2 = DecisionTreeClassifier(max_depth = md, criterion='entropy')
dec_tree_2.fit(X_train_sub, Y_train_sub)
```

Out[29]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_d
epth=7,            max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_stat
e=None,            splitter='best')
```

In [30]:

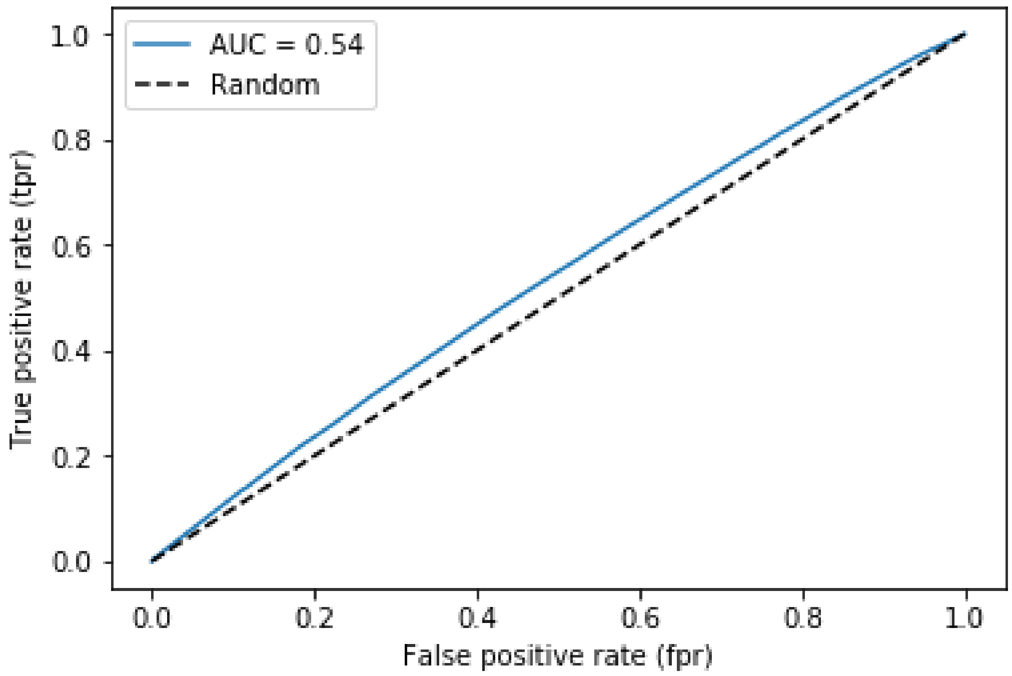
```
# Use sub test data to calculate AUC
```

```
fpr_2, tpr_2, thresholds_2 = metrics.roc_curve( Y_test_sub, dec_tree_2.predict_p
roba(X_test_sub)[:, 1] ) auc_2 = metrics.roc_auc_score(Y_test_sub,
dec_tree_2.predict_proba(X_test_sub)[: , 1]) In [31]:
```

```
plt.plot(fpr_2, tpr_2, label="AUC = %.2f" % round(auc_2, 2))
plt.xlabel("False positive rate (fpr)") plt.ylabel("True positive
rate (tpr)") plt.plot([0,1], [0,1], 'k--', label="Random")
plt.legend(loc=2)
```

Out[31]:

```
<matplotlib.legend.Legend at 0x10655ff98>
```



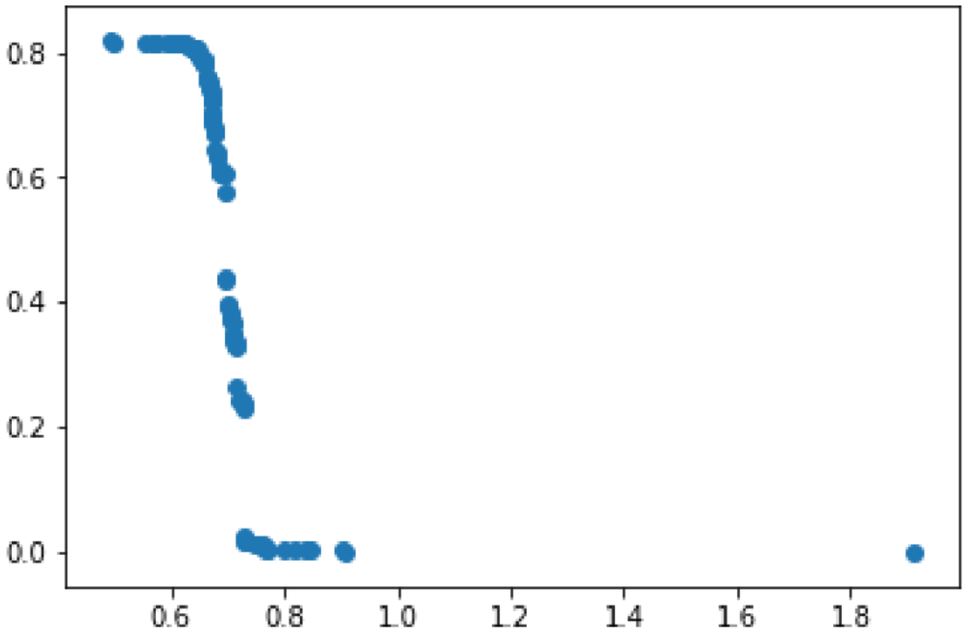
In [32]:

```
# Calculate accuracy of predictions of sub test data set based on different thre
shold values accs_2 = [] for t in thresholds_2:     acc = metrics.f1_score(
Y_test_sub, dec_tree_2.predict_proba(X_test_sub)[:,
1] > t )     accs_2.append(acc)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classificatio
n.py:1143: UndefinedMetricWarning: F-score is ill-defined and being
set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classificatio
n.py:1143: UndefinedMetricWarning: F-score is ill-defined and being
set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```

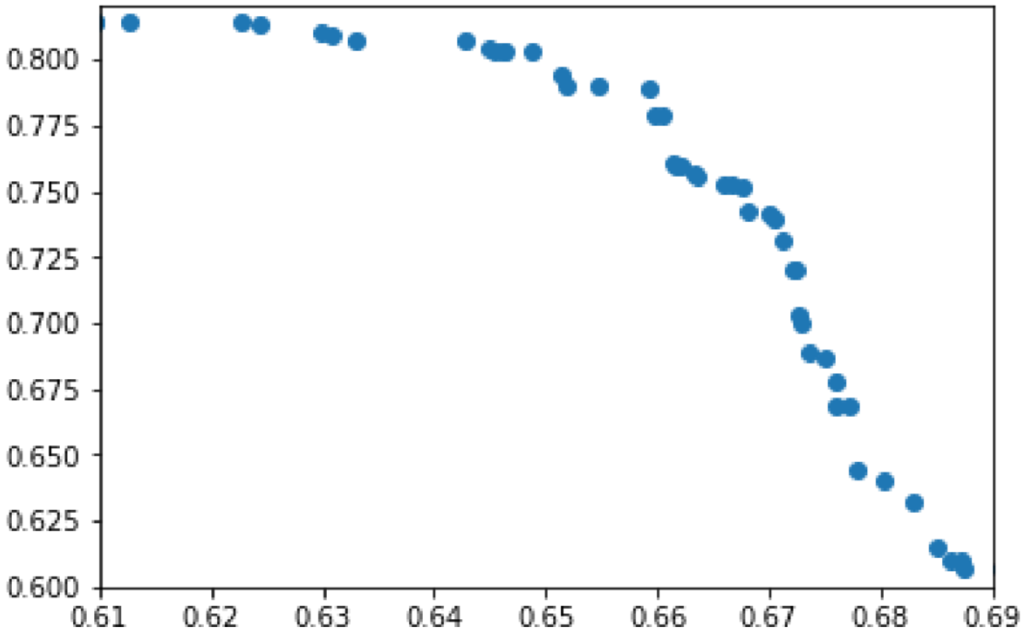
In [33]:

```
# Graph
plt.figure()
plt.scatter(thresholds_2, accs_2)
plt.show()
```



In [34]:

```
# Enlarge the graph
plt.figure()
plt.scatter(thresholds_2, accs_2)
plt.xlim(0.61, 0.69)
plt.ylim(0.6, 0.82)
plt.show()
```



From the graph, we pick 0.674 as our threshold value (the point where accuracy starts to drop sharply.

In [98]:

```
Y_test_probability_2 = dec_tree_2.predict_proba(X_test)[:, 1] > 0.673 In
```

[ ]:

In [147]:

```
model_2_c = Ridge(alpha=al_2, normalize=True)

model_2_c.fit(X_train_PC2[best_feats_2], Y_train_PC2[label2]) predictions_2_c

= model_2_c.predict(X_test_best_2)
```

In [148]:

```
predictions_2_cl = [] for i in range(len(Y_test_probability_2)):
predictions_2_cl.append((Y_test_probability_2[i]*predictions_2_c[i])[0])
predictions_2_cl = np.asarray(predictions_2_cl)
```

In [149]:

```
# Apply classification to the predictions of the ridge regression model
# If we precict a customer will buy Product_Category_2

mae_2_c = metrics.mean_absolute_error( Y_test[label2], predictions_2_cl )
predict_2_c = np.sum(predictions_2_a) actual_2_c = np.sum(Y_test[label2])[0]
off_2_c = (predict_2_c-actual_2_c)/actual_2_c

print("Product_Category_2 - after classification") print("Model:
Ridge Regression")
print("Mean absolute error: ", round(mae_2_c,4)) print("Our predictions
are",round(off_2_c*100,4),"% off from the actual values")
```

```
Product_Category_2 - after classification
Model: Ridge Regression
Mean absolute error:  6.3828
Our predictions are -2.9521 % off from the actual values
```

We get a greater mean absolute error here. This is probably because when we multiply the predictions from ridge and classification, we have a greater error when the predictions from classification go wrong.

Product_Category_3

Ridge Regression

In [154]:

```
# look at the relationship between features & Product_Category_3

# Select the best features to work with
known_good_feats = []
score = 100000

best_score_3 = 10000
best_feats_3 = []
al_3 = 0
for i in range(15):
    known_good_feats, score, a = check_next_subset( X_train, Y_train[label3], kn
own_good_feats )
    if score < best_score_3:
        best_score_3 = score
        best_feats_3 = known_good_feats
        al_3 = a
    else:
        break
print("We select",best_feats_3)
print("The best mean absolute error we get is", round(best_score_3,4))
```

```
We select ['City_Category_C', 'Gender_F', 'City_Category_A', 'Age_55
+', 'Age_46-50', 'Age_51-55', 'Age_36-45', 'Age_18-25']
The best mean absolute error we get is 5.3638
```

In [155]:

```
# set X to the selected best feats
X_train_best_3 = X_train[best_feats_3]
X_test_best_3 = X_test[best_feats_3]

# model
model_3 = Ridge(alpha=al_3, normalize=True) model_3.fit(X_train_best_3,
Y_train[label3])

predictions_3 = model_3.predict(X_test_best_3) mae_3 =
metrics.mean_absolute_error( Y_test[label3], predictions_3 )
predict_3 = np.sum(predictions_3) actual_3 =
np.sum(Y_test[label3])[0] off_3 = (predict_3-actual_3)/actual_3
print("Product_Category_3 - before classification")
print("Model: Ridge Regression")
print("Mean absolute error: ", round(mae_3,4)) print("Our predictions
are",round(off_3*100,4),"% off from the actual values")
```

```
Product_Category_3 - before classification
Model: Ridge Regression
```

```
Mean absolute error:  5.3498
Our predictions are 0.3553 % off from the actual values
```

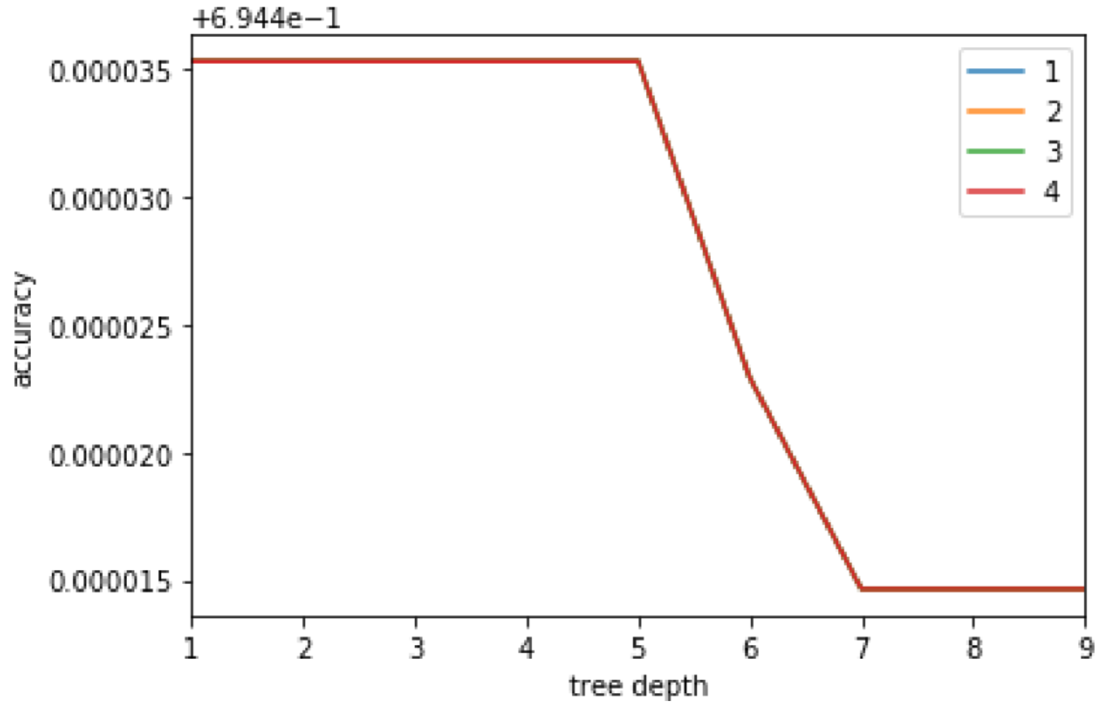
Classification

In [156]:

```python
df_3 = pd.DataFrame(index=depths_list ,columns=min_leaf_size_list)

for v in min_leaf_size_list:
    l_3 = []
    for x in depths_list:
        l_3.append(cv_eval(X_train, Y_train['Product_Category_3_1'], x, v))
    df_3[v] = l_3

ax = df_3.plot()
ax.set_xlabel("tree depth")
ax.set_ylabel("accuracy")
```

Out[156]:

```
Text(0, 0.5, 'accuracy')
```



In [159]:

```python
# Get the max depth from the maximum value of accuracy
md_3 = 5
```

In [160]:

```python
X_train_sub_3, X_test_sub_3, Y_train_sub_3, Y_test_sub_3 = train_test_split(
X_train, Y_train['Product_Category_3_1'], test_size=0.2)
```

In [161]:

```python
# Decision Tree Classifier - whether a customer will buy Product_Category_2
dec_tree_3 = DecisionTreeClassifier(max_depth = md_3, criterion='entropy')
dec_tree_3.fit(X_train_sub_3, Y_train_sub_3)
```

Out[161]:

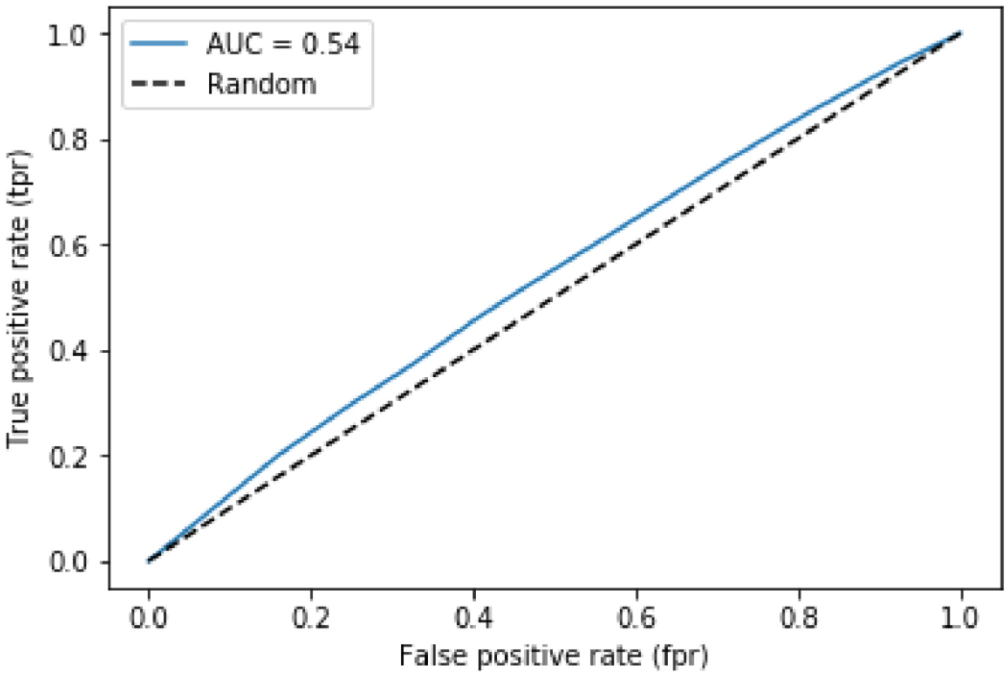
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_d
epth=5,
            max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_stat e=None,
splitter='best')
```

In [162]:

```
# Use sub test data to calculate AUC
fpr_3, tpr_3, thresholds_3 = metrics.roc_curve( Y_test_sub_3, dec_tree_3.predict
_proba(X_test_sub_3)[:, 1] ) auc_3 = metrics.roc_auc_score(Y_test_sub_3,
dec_tree_3.predict_proba(X_test_sub_
3)[:, 1])

plt.plot(fpr_3, tpr_3, label="AUC = %.2f" % round(auc_3, 2))
plt.xlabel("False positive rate (fpr)") plt.ylabel("True positive
rate (tpr)") plt.plot([0,1], [0,1], 'k--', label="Random")
plt.legend(loc=2)
```

Out[162]:

```
<matplotlib.legend.Legend at 0x1a18354c50>
```

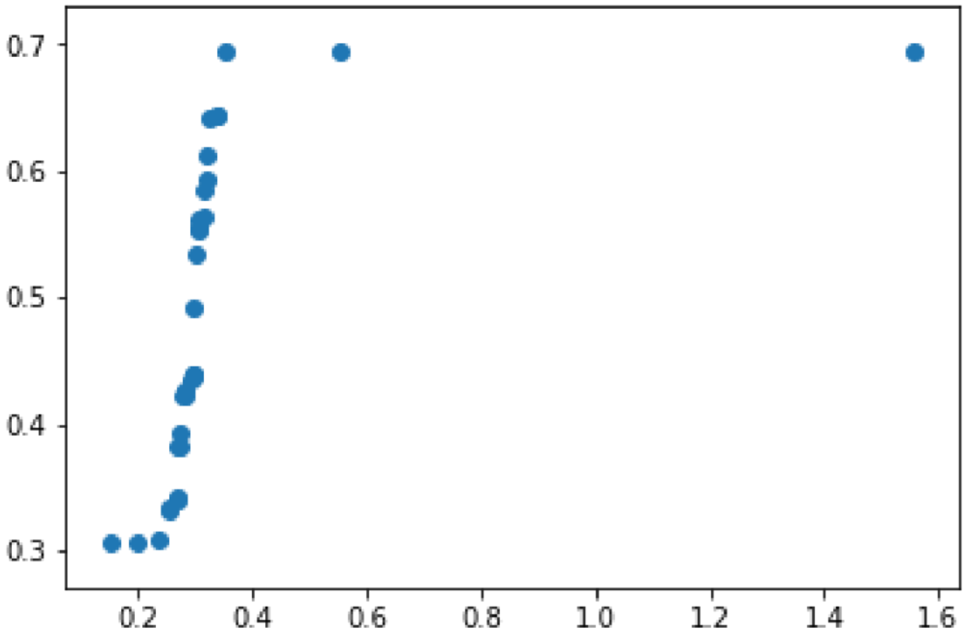


In [163]:

```
# Calculate accuracy of predictions of sub test data set based on different thre
shold values
accs_3 = []
for t in thresholds_3:
    acc = metrics.accuracy_score( Y_test_sub_3, dec_tree_3.predict_proba(X_test_
sub_3)[:, 1] > t )
    accs_3.append(acc)
```

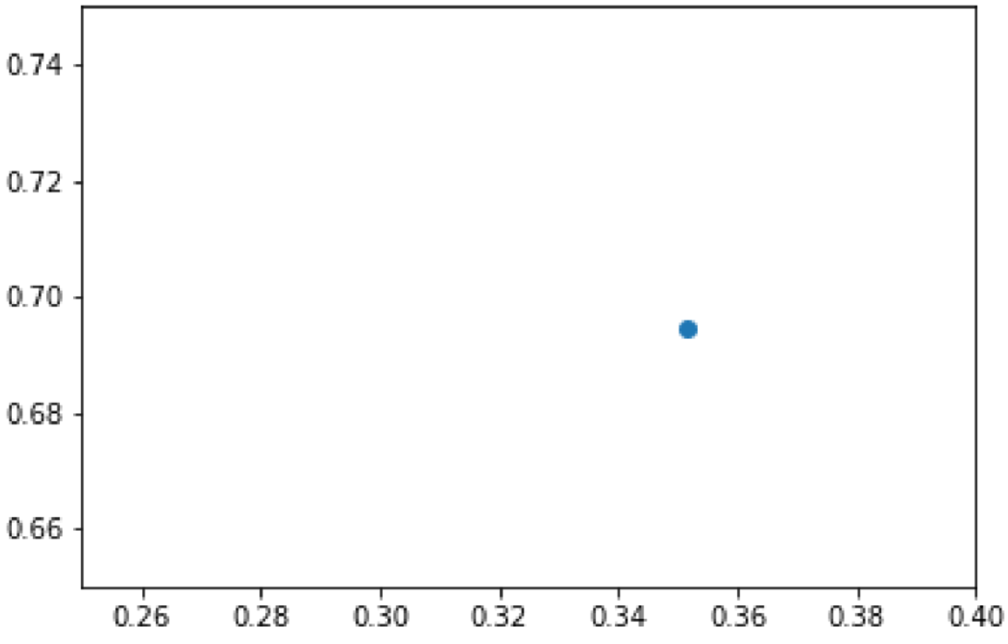
In [164]:

```
# Graph
plt.figure()
plt.scatter(thresholds_3, accs_3)
plt.show()
```



In [168]:

```
# Enlarge the graph
plt.figure()
plt.scatter(thresholds_3, accs_3)
plt.xlim(0.25, 0.4)
plt.ylim(0.65, 0.75)
plt.show()
```



Set threshold = 0.332778

In [173]:

```
Y_test_probability_3 = dec_tree_3.predict_proba(X_test)[:, 1] > 0.35 In [174]:
```

```
model_3_c = Ridge(alpha=al_3, normalize=True)

model_3_c.fit(X_train_PC3[best_feats_3], Y_train_PC3[label3]) predictions_3_c =

model_3_c.predict(X_test_best_3)

predictions_3_cl = [] for i in range(len(Y_test_probability_3)):
predictions_3_cl.append((Y_test_probability_3[i]*predictions_3_c[i])[0])
predictions_3_cl = np.asarray(predictions_3_cl)

mae_3_c = metrics.mean_absolute_error( Y_test[label3], predictions_3_cl )
predict_3_c = np.sum(predictions_3_a) actual_3_c = np.sum(Y_test[label3])[0]
off_3_c = (predict_2_c-actual_3_c)/actual_3_c

print("Product_Category_3 - after classification") print("Model:
Ridge Regression")
print("Mean absolute error: ", round(mae_3_c,4)) print("Our predictions
are",round(off_3_c*100,4),"% off from the actual values")
```

```
Product_Category_3 - after classification
Model: Ridge Regression
Mean absolute error:  4.7526
Our predictions are 71.4929 % off from the actual values
```