Foundations of Data Science Fall 2022 - Homework 2 (30 points)

Student Name: Yamini Lakshmi Narasimhan

Student Net Id: yl9822

Part 1: Preparing a Training Set and Training a Decision Tree (10 Points)

This is a hands-on task where we build a predictive model using Decision Trees discussed in class. For this part, we will be using the data in `cell2cell_data.csv` (you can find this on NYU Brightspace).

These historical data consist of 39,859 customers: 19,901 customers that churned (i.e., left the company) and 19,958 that did not churn (see the `"churndep"` variable). Here are the data set's 11 possible predictor variables for churning behavior:

```
Pos.  Var. Name  Var. Description
----- ---------- --------------------------------------------
------------------
1     revenue    Mean monthly revenue in dollars
2     outcalls   Mean number of outbound voice calls
3     incalls    Mean number of inbound voice calls
4     months     Months in Service
5     eqpdays    Number of days the customer has had his/her
current equipment
6     webcap     Handset is web capable
7     marryyes   Married (1=Yes; 0=No)
8     travel     Has traveled to non-US country (1=Yes; 0=No)
9     pcown      Owns a personal computer (1=Yes; 0=No)
10    creditcd   Possesses a credit card (1=Yes; 0=No)
11    retcalls   Number of calls previously made to retention
team
```

The 12th column, the dependent variable `"churndep"`, equals 1 if the customer churned, and 0 otherwise.

In [202…

```
import warnings
from pprint import pprint
warnings.filterwarnings('ignore')
warnings.filterwarnings(action='once')
```

1. Load the data and prepare it for modeling. Note that the features are already processed for you, so the only thing needed here is split the data into training and testing. Use pandas to create two data frames: train_df and test_df, where train_df has 80% of the data chosen uniformly at random without replacement (test_df should have the other 20%). Also, make sure to write your own code to do the splits. You may use any random() function numpy but do not use the data splitting functions from Sklearn.

(2 Points)

In [165…

```
# Place your code here
import pandas as pd
import numpy as np

df = pd.read_csv("./cell2cell_data.csv")

print("Total data",len(df))
percent = np.random.rand(len(df)) <= 0.8
train_df = df[percent]
test_df = df[~percent]

print("Train df",len(train_df))
print("Test df",len(test_df))

train_X = train_df.loc[:, train_df.columns != "churndep"]
train_Y = train_df["churndep"]


test_X = test_df.loc[:, test_df.columns != "churndep"]
test_Y = test_df["churndep"]
```

```
Total data 39833
Train df 31910
Test df 7923
```

In [166…

```
print(train_df["churndep"].value_counts())
print(test_df["churndep"].value_counts())
```

```
0    15985
1    15925
Name: churndep, dtype: int64
0    3965
1    3958
Name: churndep, dtype: int64
```

In [167…

```
df.head()
```

Out[167]:

	revenue	outcalls	incalls	months	eqpdays	webcap	marryyes	travel	pcown	creditcc
0	48.82	10.00	3.0	26	780	0	0	0	0	
1	83.53	20.00	1.0	31	745	1	0	0	0	
2	29.99	0.00	0.0	52	1441	0	0	0	1	
3	51.42	0.00	0.0	36	59	1	0	0	0	
4	37.75	2.67	0.0	25	572	0	0	0	1	

2. Now build and train a decision tree classifier using `DecisionTreeClassifier()` (manual page) on train_df to predict the `"churndep"` target variable. Make sure to use `criterion='entropy'` when instantiating an instance of `DecisionTreeClassifier()`. For all other settings you should use all of the default options.

(1 Point)

In [168…

```
# Place your code here
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(criterion='entropy')
dtc.fit(train_X, train_Y)
```

Out[168]:

```
▾            DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

3. Using the resulting model from 1.2, show a bar plot of feature names and their feature importance (hint: check the attributes of the `DecisionTreeClassifier()` object directly in IPython or check the manual!).

(3 Points)

In [169…

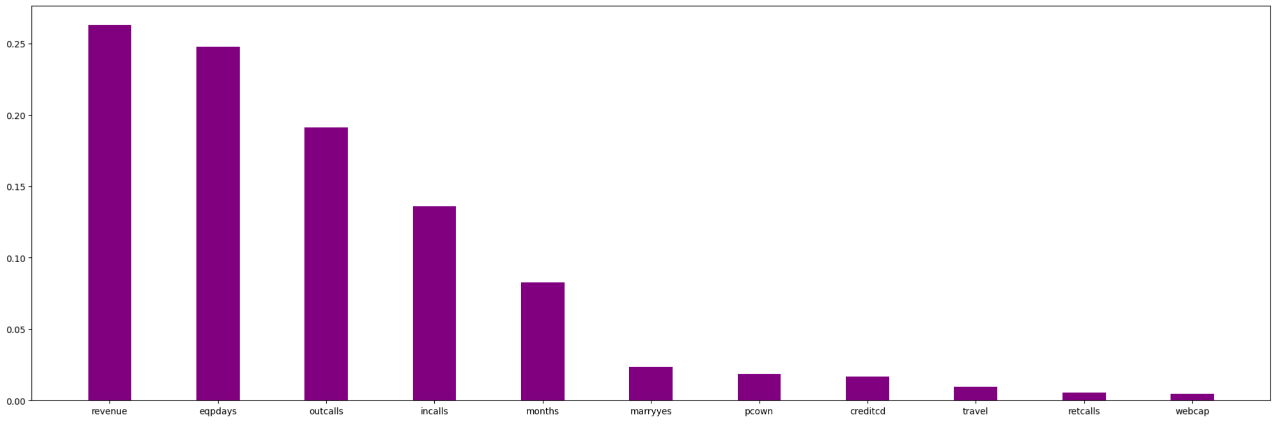
```python
# Place your code here
import matplotlib.pyplot as plt
feature_importance = {}
for importance, name in sorted(zip(dtc.feature_importances_, train_X),revers
    feature_importance[name]=importance

plt.figure(figsize=(25,8))
print(feature_importance)
plt.bar(feature_importance.keys(), feature_importance.values(), color ='purp
```

```
{'revenue': 0.2633111890880889, 'eqpdays': 0.2477853547757061, 'outcalls':
0.19113101270417757, 'incalls': 0.13606175408566842, 'months': 0.0828986580
2309869, 'marryyes': 0.02353186425352615, 'pcown': 0.01855427418985122, 'cr
editcd': 0.01671943163250753, 'travel': 0.009737491800810492, 'retcalls': 0
.005654106620520814, 'webcap': 0.004614862826043981}
```

Out[169]:

```
<BarContainer object of 11 artists>
```



4. Is the relationship between the top 3 most important features (as measured here) negative or positive? If your marketing director asked you to explain the top 3 drivers of churn, how would you interpret the relationship between these 3 features and the churn outcome? What "real-life" connection can you draw between each variable and churn?

(2 Points)

In [112…

```python
df_corr = df[["revenue","eqpdays", "outcalls","churndep"]]
df_corr.corr()
```

Out[112]:

	revenue	eqpdays	outcalls	churndep
revenue	1.000000	-0.222074	0.500709	-0.013370
eqpdays	-0.222074	1.000000	-0.244112	0.112821
outcalls	0.500709	-0.244112	1.000000	-0.037071
churndep	-0.013370	0.112821	-0.037071	1.000000

Top 3 drivers of churn are revenue, outcalls and eqpdays of the company revenue <-> churndep -0.013370 negative outcalls<-> churndep -0.037071 negative eqpdays <-> churndep +0.112821 positive

Revenue is negatively correlated that means higher the revenue(Mean monthly revenue in dollars) lower the churndep, similarly outcalls and churndep are also negatively correlated, eqpdays and churndep are positively correlated that is higher eqpdays then higher churndep

Revenue: Revenue is negatively correlated that means higher the revenue lower the churndep It makes sense that if the revenue is high for the company then churndep is low as the clients will trust the company lot more since profits are flowing in Another relation could be that if revenue is flowing in it means clients are flushing the money in so high revenue means low number of clients are leaving and lot of clients are currently paying

Outcalls: Outcalls being negatively correlated also makes sense as it means Mean number of outbound voice calls with the client is lot so client feels a lot more reliable of the company so churndep is lower.

Eqpdays: Number of days the customer has had his/her current equipment : positively correlated which means higher the no of days the customer has their equipment, higher the churndep. May be the equipment becomes faulty after few days and there are no proper customer service because of which the customer tends to leave.

5. Using the classifier built in 1.2, try predicting `"churndep"` on both the train_df and test_df data sets. What is the accuracy on each?

(2 Points)

In [170…

```
# Place your code here
# Place your code here
from sklearn.metrics import accuracy_score
predtest_Y = dtc.predict(test_X)
print("Test Accuracy", accuracy_score(test_Y, predtest_Y))

predtrain_Y = dtc.predict(train_X)
print("Train Accuracy", accuracy_score(train_Y, predtrain_Y))
```

```
Test Accuracy 0.5367916193361101
Train Accuracy 0.9998433093074272
```

Part 2 - Finding a Good Decision Tree (Total 10 Points)

The default options for your decision tree may not be optimal. We need to analyze whether tuning the parameters can improve the accuracy of the classifier. For the following options `min_samples_split` and `min_samples_leaf`:

1. Generate a list of 10 values of each for the parameters min_samples_split and min_samples_leaf.

(1 Point)

```
min_samples_leaf: int, float, optional (default=1)
The minimum number of samples required to be at a leaf node.
A split point at any depth will only be considered if it
leaves at least min_samples_leaf training samples in each of
the left and right branches. This may have the effect of
smoothing the model, especially in regression.
min_samples_leaf is also used to control over-fitting by
defining that each leaf has more than one element. Thus
ensuring that the tree cannot overfit the training dataset by
creating a bunch of small branches exclusively for one sample
each. In reality, what this is actually doing is simply just
telling the tree that each leaf doesn’t have to have an
impurity of 0

min_samples_split: int or float, default=2
The minimum number of samples required to split an internal
node:
If int, then consider min_samples_split as the minimum
number.
If float, then min_samples_split is a fraction and
ceil(min_samples_split * n_samples) are the minimum number of
samples for each split.
```

2. Explain in words your reasoning for choosing the above ranges.

```
min_samples_leaf
Chose a range between 100-1750 with a difference of 250
because since we have a dataset of 22k and only 2 classes to
predict with only 5 features that contribute to the
predictions - revenue, eqpdays, outcalls, incalls and months
- the child nodes definitely can have a minimum of range 100
and can go higher as the dataset will be majorly split by 5
features split - this will also prevent overfitting and
generalising the model

min_samples_split
Chose the same range as we don't want the dataset to overfit
on the training sample and having the less than 100 would not
only overfit but will also create a complex tree with
multiple nodes and for 5 contributing features a shallow tree
would be sufficient with multiple samples grouped together in
a node
```

Place your response here

3. For each combination of values in 3.1 (there should be 100), build a new classifier and check the classifier's accuracy on the test data. Plot the test set accuracy for these options. Use the values of `min_samples_split` as the x-axis and generate a new series (line) for each of `min_samples_leaf`.

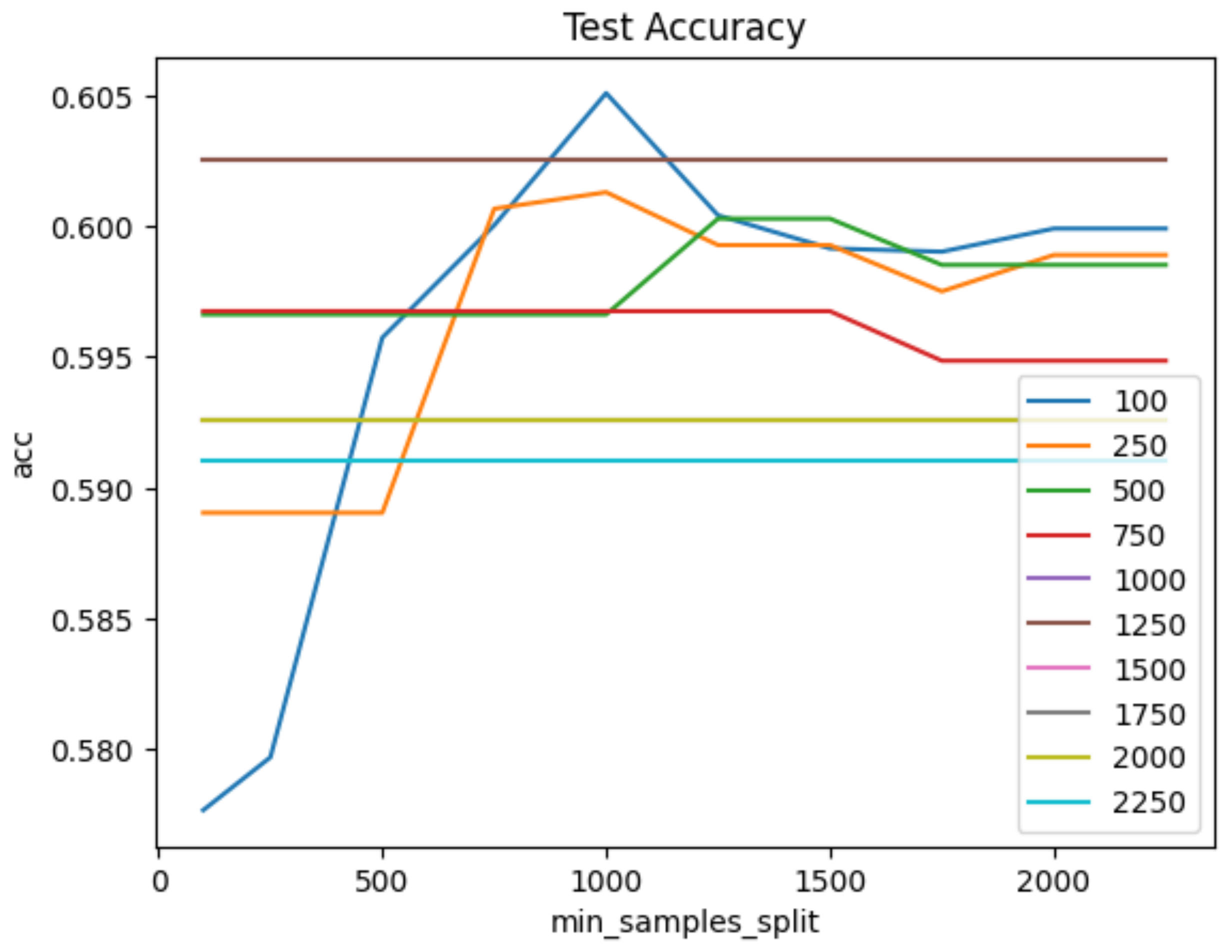
(5 Points)

```
In [264…  # Place your code here
          min_samples_split = [100, 250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250]
          min_samples_leaf =[100, 250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250]
          j=0


          for i in range(len(min_samples_leaf)):
              acc_lis = []
              for j in range(len(min_samples_split)):
                  dtc = DecisionTreeClassifier(random_state=0, min_samples_split=min_s
                  predtest_Y = dtc.predict(test_X)
                  acc = accuracy_score(test_Y, predtest_Y)
                  acc_lis.append(acc)

              line = plt.plot(min_samples_split, acc_lis, label = str(min_samples_leaf

          plt.xlabel("min_samples_split")
          plt.ylabel("acc")
          plt.legend()
          plt.title("Test Accuracy")
          plt.show()
```



4. Which configuration returns the best accuracy? What is this accuracy? (Note, if you don't see much variation in the test set accuracy across values of min_samples_split or min_samples_leaf, try redoing the above steps with a different range of values).

(1 Point)

Best configuration is when min_samples_split is 1000 and min_samples_leaf is 100 and accuracy is around 60.5%

Place your response here

5. If you were working for a marketing department, how would you use your churn production model in a real business environment? Explain why churn prediction might be good for the business and how one might improve churn by using this model.

(2 Points)

I would use this model to periodically check which customer/client might have a tendency to get churned and take precautionary measure beforehand. For example if the model predicts a client/customer is going to leave and then we can always increase outbound and inbound calls as that might make the customer feel they're appreciated and stay with the company for longer or bring in more revenue by transparently showing them how we can improve

I can change my focus to companies that have a greater tendency to drop working with us there by giving them a better service and better deal for them to stay with us longer

Place your response here

Part 3: Model selection with cross-validation (5 points)

In this part, we will focus on cross-validation to find a good value for parameter `max_depth`.

1. Write a cross-validation function that does the following:

- Takes as inputs a dataset, a label name, # of splits/folds (`k`), and a sequence of values for the maximum depth of the tree (`max_depth`).
- Shuffles the data.
- Splits the data into `k` folds according to the cross-validation logic
- Performs two loops
    - Outer Loop: `for each f in range(k)` :
        - Inner Loop: `for each value in max_depth_sequence` :
            - Trains a Decision Tree on the training split with the `max_depth=value` (USE criterion='entropy' BUT DO NOT ALTER THE OTHER PARAMETERS)
            - Computes accuracy_value_f on test split
            - Stores accuracy_value_f in a dictionary of values
- Returns a dictionary, where each key-value pair is: `value: [accuracy_value_1,...,accuracy_value_k]`

(2 Points)

In [210…

```
from random import randrange

def xValDecisionTree(dataset, label, k, max_depth_sequence):
    dataset = dataset.sample(frac=1).reset_index(drop=True)
    df_split = np.array_split(dataset, k)
    train_split = pd.DataFrame()
    acc_dict = {}
    for i in range(0,k):
        test_split = pd.DataFrame(df_split[i])
        for j in range(k):
            if i!= j:
                train_split = pd.concat([pd.DataFrame(df_split[j]), train_sp
        acc_lis = []
        depth_lis = []

        train_X = train_split.loc[:, train_split.columns != label]
        train_Y = train_split[label]


        test_X = test_split.loc[:, test_split.columns != label]
        test_Y = test_split[label]


        for value in max_depth_sequence:
            dtc = DecisionTreeClassifier(random_state=0, max_depth=value,  c
            predtest_Y = dtc.predict(test_X)
            acc = accuracy_score(test_Y, predtest_Y)
            acc_lis.append(acc)
            depth_lis.append(value)
            if value not in acc_dict.keys():
                acc_dict[value] = []
            acc_dict[value].append(acc)
        plt.show()

    return acc_dict
```

2. Using the function written above, do the following:

- Generate a sequence `max_depth_sequence = [None, 2, 4, 8, 16, 32, 128, 256, 512]` (Note that None is the default value for this parameter).

2. Call accs = xValDecisionTree(dataset, 'churndep', 10, `max_depth_sequence`)
3. For each value in accs.keys(), calculate mean(accs[value]). What value is associated with the highest accuracy mean?
4. For each value in accs.keys(), calculate the ranges mean(accs[value]) +/- std(accs[value]). Do the ranges associated with the value that has the highest mean(accs[value]) overlap with ranges for other values? What may this suggest and what are the limitations of a standard deviation based analysis in this scenario?
5. Which depth value would you pick, if any, and why?

(3 Points)

In [257…
```
# Place your code here
#2.
max_depth_sequence = [None, 2, 4, 8, 16, 32, 128, 256, 512]
accs_dict = xValDecisionTree(df, "churndep", 10,  max_depth_sequence)
pprint(accs_dict)
```

```
{None: [0.5348895582329317,
        0.9997489959839357,
        1.0,
        0.9992467988953051,
        0.9997489329651017,
        0.9997489329651017,
        0.9997489329651017,
        0.9997489329651017,
        1.0,
        0.9997489329651017],
 2: [0.5820783132530121,
     0.5747991967871486,
     0.5860943775100401,
     0.56866683404469,
     0.5849861913130806,
     0.5824755209640974,
     0.5827265879989957,
     0.5895053979412503,
     0.5827265879989957,
     0.590760733115742],
 4: [0.5978915662650602,
     0.5956325301204819,
     0.5938755020080321,
     0.5842329902083856,
     0.5985438111975897,
     0.5872457946271654,
     0.5992970123022847,
     0.6055736881747427,
     0.6008034145116746,
     0.5967863419533015],
 8: [0.5946285140562249,
     0.6056726907630522,
     0.6054216867469879,
     0.6038162189304545,
     0.6136078332914888,
     0.6010544815465729,
     0.6171227717800652,
     0.6148631684659804,
     0.6211398443384384,
     0.6048204870700477],
 16: [0.5662650602409639,
      0.6859939759036144,
      0.696285140562249,
      0.6921918152146623,
      0.6869194074817977,
      0.6768767260858649,
      0.6778809942254582,
      0.7102686417273412,
      0.7062515691689681,
      0.7072558373085613],
 32: [0.5431726907630522,
      0.9427710843373494,
      0.9492971887550201,
      0.9540547326136078,
      0.9450163193572684,
      0.950288727090133,
      0.9480291237760482,
      0.9422545819733869,
      0.9590760733115742,
      0.9477780567411499],
 128: [0.5348895582329317,
       0.9997489959839357,
       1.0,
       0.9992467988953051,
       0.9997489329651017,
       0.9997489329651017,
       0.9997489329651017,
       0.9997489329651017,
       1.0,
       0.9997489329651017],
 256: [0.5348895582329317,
       0.9997489959839357,
       1.0,
       0.9992467988953051,
       0.9997489329651017,
       0.9997489329651017,
       0.9997489329651017,
       0.9997489329651017,
       1.0,
       0.9997489329651017],
 512: [0.5348895582329317,
       0.9997489959839357,
       1.0,
       0.9992467988953051,
       0.9997489329651017,
       0.9997489329651017,
       0.9997489329651017,
       0.9997489329651017,
       1.0,
       0.9997489329651017]}
```

In [214…

```
accs_df = pd.DataFrame.from_dict(accs_dict)
accs_df
```

Out[214]:

	NaN	2.0	4.0	8.0	16.0	32.0	128.0	256.0	
0	0.533133	0.577309	0.586847	0.581576	0.565261	0.535392	0.533133	0.533133	0.5
1	0.999749	0.590110	0.604418	0.609940	0.694528	0.948042	0.999749	0.999749	0.9
2	0.999749	0.581576	0.594378	0.601155	0.693273	0.943775	0.999749	0.999749	0.9
3	0.999749	0.582224	0.596033	0.608838	0.684911	0.934220	0.999749	0.999749	0.9
4	0.998745	0.581973	0.595280	0.607331	0.687170	0.934220	0.998745	0.998745	0.9
5	1.000000	0.576199	0.592769	0.607080	0.691690	0.949033	1.000000	1.000000	1.0
6	0.999749	0.590008	0.603314	0.620889	0.697464	0.950540	0.999749	0.999749	0.9
7	0.999749	0.576450	0.591765	0.607582	0.693698	0.939744	0.999749	0.999749	0.9
8	1.000000	0.586744	0.600803	0.618880	0.712779	0.947527	1.000000	1.000000	1.0
9	0.999498	0.580216	0.590761	0.603565	0.696711	0.945518	0.999498	0.999498	0.9

The maximum accuracy in the val set is 0.999498 when max_depth = None, 128, 256, 512

In [256…

```
#3
describe_df = accs_df.describe()
describe_df.loc['mean-std'] = describe_df.loc['mean'] - describe_df.loc['st
describe_df.loc['mean+std'] = describe_df.loc['mean'] + describe_df.loc['st
describe_df
```

Out[256]:

	NaN	2.0	4.0	8.0	16.0	32.0	128.0
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	0.953012	0.582281	0.595637	0.606684	0.681749	0.902801	0.953012
std	0.147531	0.005180	0.005665	0.010739	0.041607	0.129227	0.147531
min	0.533133	0.576199	0.586847	0.581576	0.565261	0.535392	0.533133
25%	0.999561	0.578036	0.592016	0.604444	0.688300	0.935601	0.999561
50%	0.999749	0.581775	0.594829	0.607457	0.693486	0.944647	0.999749
75%	0.999749	0.585614	0.599611	0.609664	0.696165	0.947913	0.999749
max	1.000000	0.590110	0.604418	0.620889	0.712779	0.950540	1.000000
mean-std	0.805481	0.577101	0.589972	0.595945	0.640142	0.773574	0.805481
mean+std	1.100543	0.587461	0.601302	0.617422	0.723356	1.032028	1.100543

4. Standard deviation here shows how the data in folds are distributed that is if the std of accuracy is low then it means that the data is distributed uniformly among the folds whereas if std is high then it shows that the data in the val fold is highly variable and is not similar to the training data folds

5. The range(mean - std to mean + std) is the same for depth = None, 256 and 512 its the same range that is 0.805481 - 1.100543

It means the data is overfitting to the train data so the val set accuracy is extremely high So, i'll probably take the next best accuracy that is with value 16.0 as max_depth_sequence

In [258…

```
dtc = DecisionTreeClassifier(random_state=0, max_depth=None,  criterion='ent
predtest_Y = dtc.predict(test_X)
acc = accuracy_score(test_Y, predtest_Y)
print("For max_depth = None Acc = ",acc)


dtc = DecisionTreeClassifier(random_state=0, max_depth=256,  criterion='entr
predtest_Y = dtc.predict(test_X)
acc = accuracy_score(test_Y, predtest_Y)
print("For max_depth = 256 Acc = ",acc)

dtc = DecisionTreeClassifier(random_state=0, max_depth=512,  criterion='entr
predtest_Y = dtc.predict(test_X)
acc = accuracy_score(test_Y, predtest_Y)
print("For max_depth = 512 Acc = ",acc)
```

```
For max_depth = None Acc =  0.5386848416004039
For max_depth = 256 Acc =  0.5386848416004039
For max_depth = 512 Acc =  0.5386848416004039
```

This shows how the test data is not performing that great on the best performing model in the train data as those iteration are over fitting, whereas the one with max_depth of 16 has generalised well so performs better than max_depth = None, 128, 256, 512

In [259…

```
dtc = DecisionTreeClassifier(random_state=0, max_depth=16,  criterion='entro
predtest_Y = dtc.predict(test_X)
acc = accuracy_score(test_Y, predtest_Y)
print("For max_depth = 16 Acc = ",acc)
```

```
For max_depth = 16 Acc =  0.566578316294333
```

In [ ]:

Part 4: Boosting (5 Points)

Now, as we covered in class, ensemble methods are often used to improve performance.

1. Implement the boosting algorithm: XGBoost for the same `cell2cell_data.csv` task as above. You will have to select how to tune hyperparameters. Besides depth, which other hyperparametrs do you optimize for? (2 points)

```
XGBoost hyperparameters

General Parameters
booster
gbtree defaut which is okay as it is used for tree based
```

xgboost

Booster Parameters
eta
It is the step size shrinkage used in update to prevent overfitting.
Range : [0,1] Typical final values : 0.01-0.2.\n
Default : 0

gamma
A node is split only when the resulting split gives a positive reduction in the loss function. The larger gamma is, the more conservative the algorithm will be.
Range: [0,∞]
Default : 0

max_depth - used 3 because mostly there are 3 major features that is eqpdays, revenue, outcalls
It is used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
range: [0,∞] (0 is only accepted in lossguided growing policy when tree_method is set as hist.
default =6

min_child_weight - for overfitting we have modified tree depth etc
It defines the minimum sum of weights of all observations required in a child.
range: [0,∞]
It is used to control over-fitting.
Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
default=1

max_delta_step - churndep is not biased(0: 15991 1: 15951) so not using this
In maximum delta step we allow each tree’s weight estimation to be.
Usually this parameter is not needed, but it might help in logistic regression when class is extremely imbalanced.
Set it to value of 1-10 might help control the update.
range: [0,∞]

```
default=0


subsample   - Want to give this default because this is too
much rules on the tree
It denotes the fraction of observations to be randomly
samples for each tree.
Subsample ratio of the training instances.
Setting it to 0.5 means that XGBoost would randomly sample
half of the training data prior to growing trees. - This will
prevent overfitting.
Typical values: 0.5-1
range: (0,1]

lambda - 1 is good enough
L2 regularization term on weights (analogous to Ridge
regression).
This is used to handle the regularization part of XGBoost.
Increasing this value will make model more conservative.
default=1


alpha - No need this because less features and we don't need
feature sleection here
L1 regularization term on weights (analogous to Lasso
regression).
It can be used in case of very high dimensionality so that
the algorithm runs faster when implemented.
Increasing this value will make model more conservative.
default=0


tree_method - auto takes care of which one to choose based on
dataset size so no need of change
Choices: auto, exact, approx, hist, gpu_hist
auto: Use heuristic to choose the fastest method.
For small to medium dataset, exact greedy (exact) will be
used.
For very large dataset, approximate algorithm (approx) will
be chosen.
Because old behavior is always use exact greedy in single
machine, user will get a message when approximate algorithm
is chosen to notify this choice.
exact: Exact greedy algorithm.
approx: Approximate greedy algorithm using quantile sketch
and gradient histogram.
hist: Fast histogram optimized approximate greedy algorithm.
```

```
It uses some performance improvements such as bins caching.
gpu_hist: GPU implementation of hist algorithm.
default= auto


scale_pos_weight - No need as classes are balanced for us
It controls the balance of positive and negative weights,
It is useful for imbalanced classes.
A value greater than 0 should be used in case of high class
imbalance as it helps in faster convergence.
A typical value to consider: sum(negative instances) /
sum(positive instances).
default=1


max_leaves  - too much interference to the tree so no need
Maximum number of nodes to be added.
Only relevant when grow_policy=lossguide is set.


Learning Task Parameters

objective  - binary:logistic because this is binary
classification
reg:logistic : logistic regression
binary:logistic : logistic regression for binary
classification, output probability
binary:logitraw: logistic regression for binary
classification, output score before logistic transformation
binary:hinge : hinge loss for binary classification. This
makes predictions of 0 or 1, rather than producing
probabilities.
multi:softmax : set XGBoost to do multiclass classification
using the softmax objective, you also need to set
num_class(number of classes)
multi:softprob : same as softmax, but output a vector of
ndata nclass, which can be further reshaped to ndata nclass
matrix. The result contains predicted probability of each
data point belonging to each class.
default=reg:squarederror

eval_metric - default metrics makes sense
The metric to be used for validation data.
The default values are rmse for regression, error for
classification and mean average precision for ranking.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Place your answer here regarding hyperparameters.

In [260…
```
import xgboost as xgb
xgb_model = xgb.XGBClassifier()
xgb_model.fit(train_X, train_Y)
y_pred = xgb_model.predict(test_X)
acc = accuracy_score(test_Y, y_pred)
print(acc)
```

```
0.5948504354411208
```

In [263…
```
import xgboost as xgb
xgb_model = xgb.XGBClassifier(booster = 'dart', eta = 0.3, max_depth = 3, ob
xgb_model.fit(train_X, train_Y)
y_pred = xgb_model.predict(test_X)
acc = accuracy_score(test_Y, y_pred)
print(acc)
```

```
0.60734570238546
```

2. Now compare the XGBoost performance to the decision tree implementation from part 3. Describe in text how they compare, and if this aligns with what you expect. (3 points)

Place your code here

```
Decision tree was over fitting for max_depth of None, 128,
256 and 512 so I took max_depth of 16 and the test set was
performing around 56.7% whereas xgboost is around 60.7%

XgBoost was bound to do better as it builds multiple trees -
Each new tree is built to improve on the deficiencies of the
previous trees and this concept is called boosting.
when compared to
decision tree that has only on tree so its deficiency lies
there with no other chance for it to change itself


XgBoost also uses gradient of the previous tree into the new
tree which helps in learning and retaining previous tree
information
unlike Decision tree that has only one tree which gives less
flexibility to learn
```

End of homework