

Project - 1

1. For criterion if no value is mentioned then by default Gini is considered.

The screenshot shows a Jupyter Notebook interface with the title "Decision Trees and kNN". The notebook content includes a section titled "Class DecisionTreeClassifier in Scikit-learn". It explains that the main parameters of the `sklearn.tree.DecisionTreeClassifier` class are:

- `max_depth` - the maximum depth of the tree;
- `criterion` - ("gini", "entropy"), default="gini"
- `max_features` - the maximum number of features with which to search for the best partition (this is necessary with a large number of features because it would be "expensive" to search for partitions for all features);
- `min_samples_leaf` - the minimum number of samples in a leaf. This parameter prevents creating trees where any leaf would have only a few members.

It then states: "The parameters of the tree need to be set depending on input data, and it is usually done by means of cross-validation, more on this later."

Next, it says: "Let's try to separate these two classes by training an Sklearn decision tree. We will use `max_depth` parameter that limits the depth of the tree. Let's visualize the resulting separating boundary."

The code cell (In [5]:) contains the following code:

```
from sklearn.tree import DecisionTreeClassifier
import pydotplus
from sklearn.tree import export_graphviz

# Let's write an auxiliary function that will return grid for further visualization.

def tree_graph_to_png(tree, feature_names, png_file_to_save):
    tree_str = export_graphviz(tree, feature_names=feature_names,
                               filled=True, out_file=None)
    graph = pydotplus.graph_from_dot_data(tree_str)
    graph.write_png(png_file_to_save)
```

For `max_depth` is None, it expands tree until all leaves are pure.

The screenshot shows a Jupyter Notebook interface with the title "Decision Trees and kNN". The notebook content includes a section titled "Let's try to separate these two classes by training an Sklearn decision tree. We will use `max_depth` parameter that limits the depth of the tree. Let's visualize the resulting separating boundary."

The code cell (In [47]:) contains the following code:

```
from sklearn.tree import DecisionTreeClassifier
import pydotplus
from sklearn.tree import export_graphviz

# Let's write an auxiliary function that will return grid for further visualization.

def tree_graph_to_png(tree, feature_names, png_file_to_save):
    tree_str = export_graphviz(tree, feature_names=feature_names,
                               filled=True, out_file=None)
    graph = pydotplus.graph_from_dot_data(tree_str)
    graph.write_png(png_file_to_save)

def get_grid(data):
    x_min, x_max = data[:, 0].min() - 1, data[:, 0].max() + 1
    y_min, y_max = data[:, 1].min() - 1, data[:, 1].max() + 1
    return np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

# For Assignment you have to change max_depth parameter and re-run the cell
clf_tree = DecisionTreeClassifier(max_depth=None, random_state=17)

# training the tree
clf_tree.fit(train_data, train_labels)

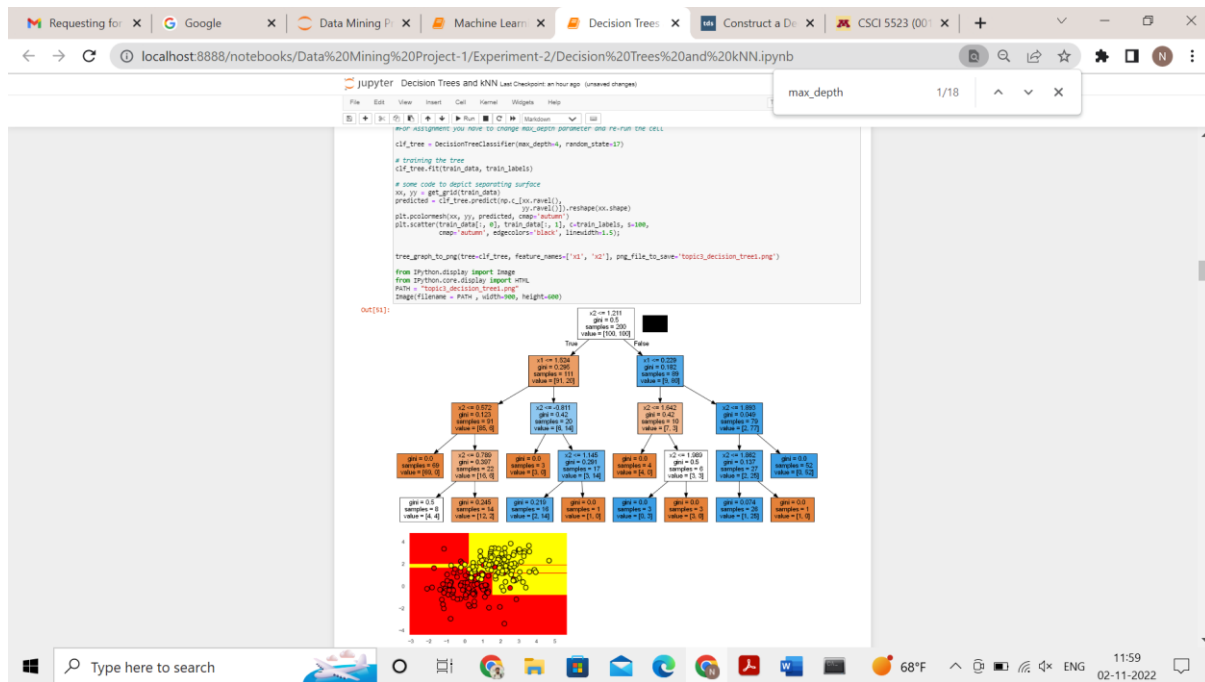
# some code to depict separating surface
xx, yy = get_grid(train_data)
predicted = clf_tree.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
plt.pcolormesh(xx, yy, predicted, cmap='autumn')
plt.scatter(train_data[:, 0], train_data[:, 1], c=train_labels, s=100,
            cmap='autumn', edgecolors='black', linewidth=1.5)

tree_graph_to_png(tree=clf_tree, feature_names=['x1', 'x2'], png_file_to_save='topic3_decision_tree1.png')

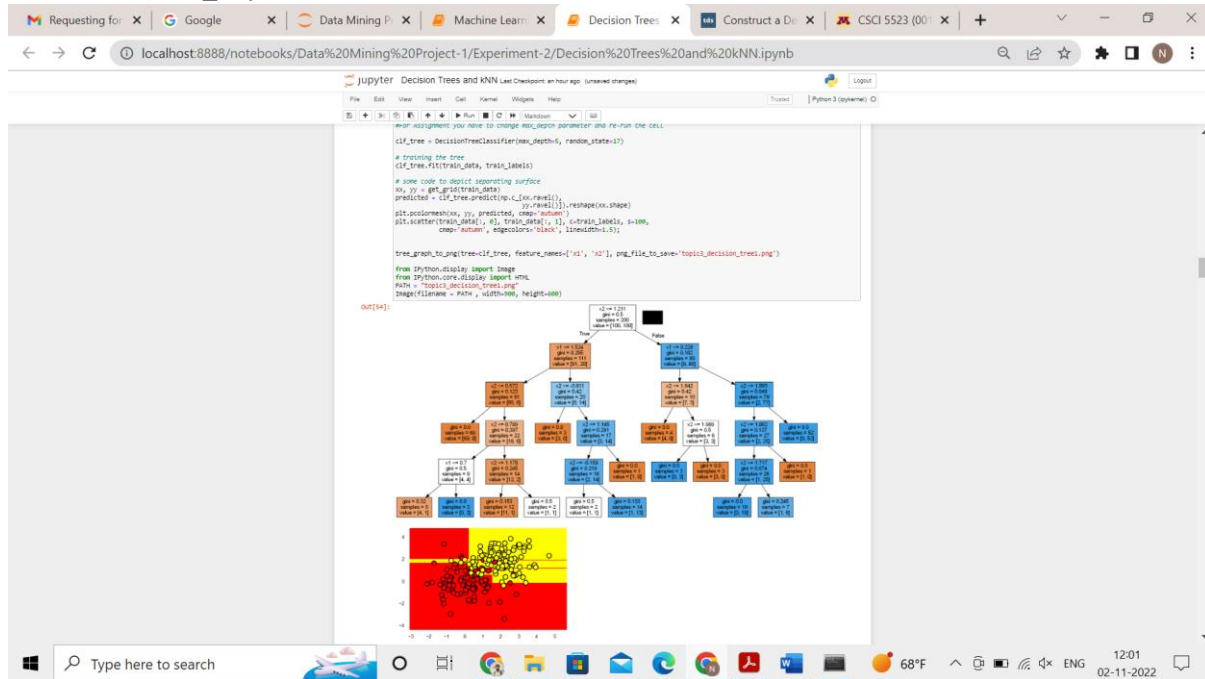
from IPython.display import Image
from IPython.core.display import HTML
PATH = "topic3_decision_tree1.png"
Image(filename = PATH , width=900, height=600)
```

2. For Overfit (`max_depth=4`)

Project - 1

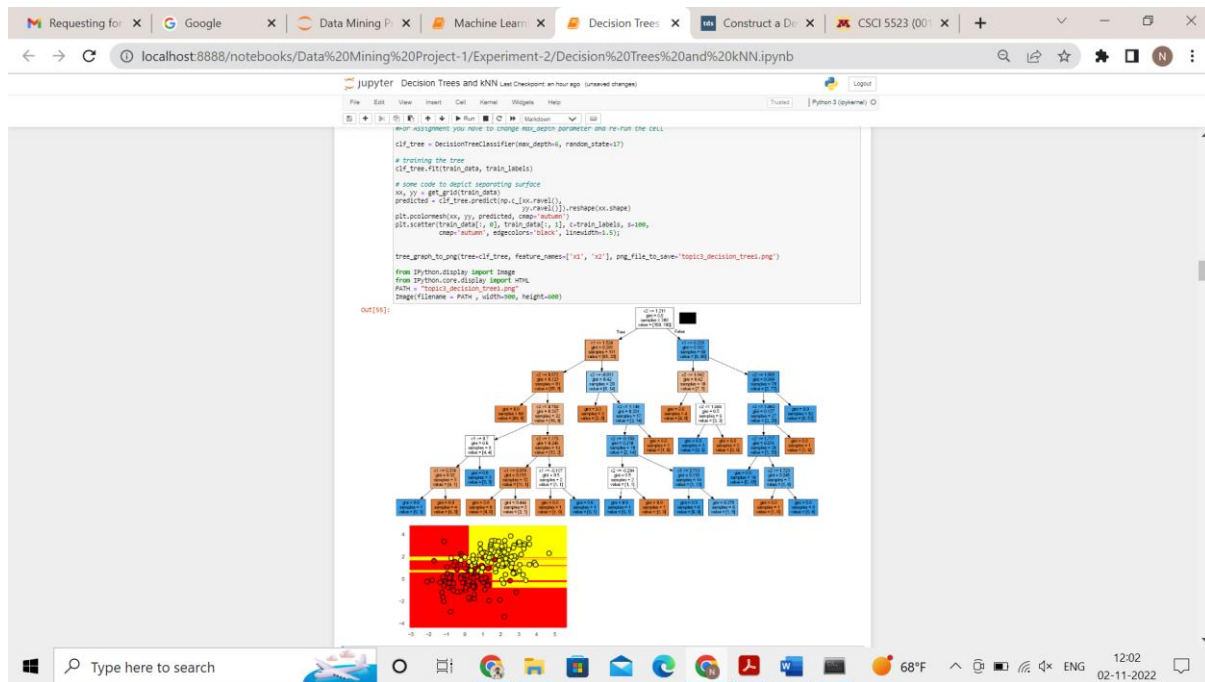


For Overfit (max_depth=5)



For Overfit (max_depth=6)

Project - 1

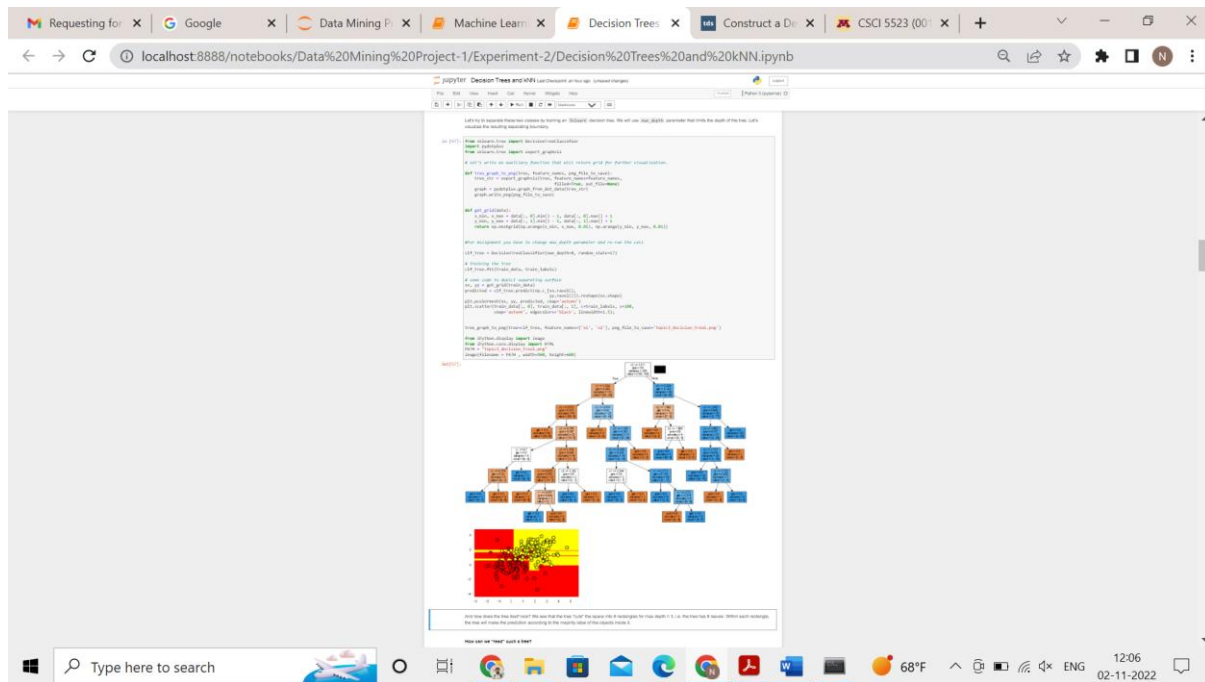


For Overfit (max_depth=7)

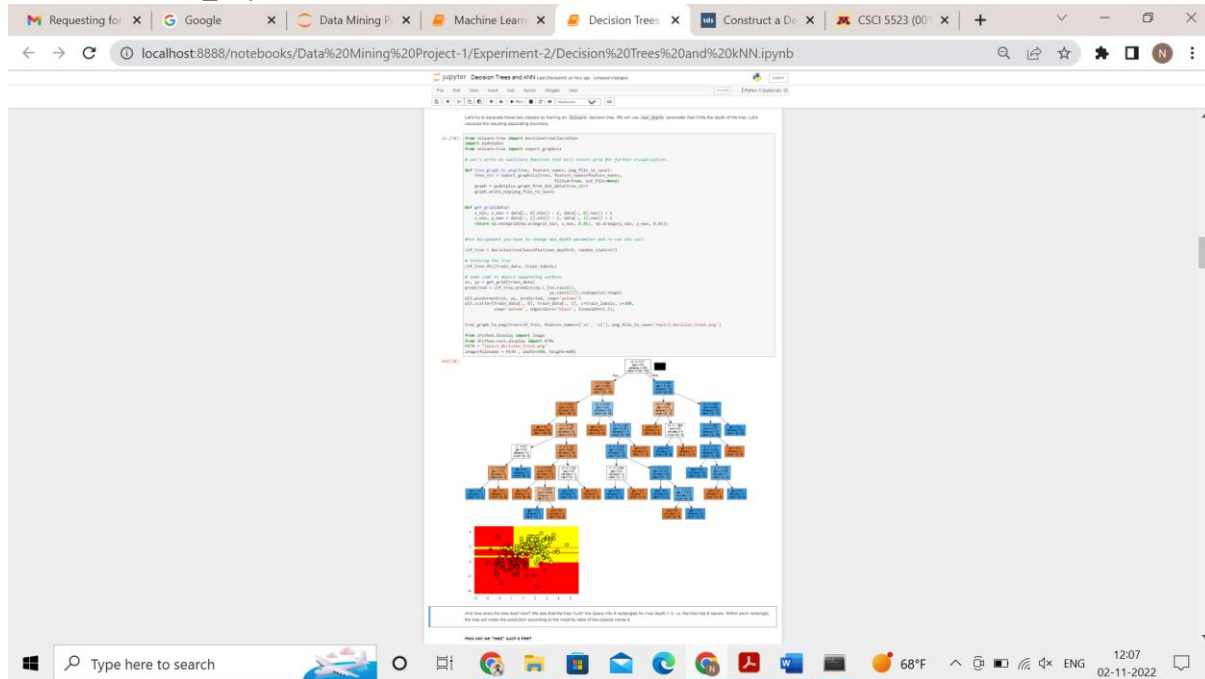


For Overfit (max_depth=8)

Project - 1

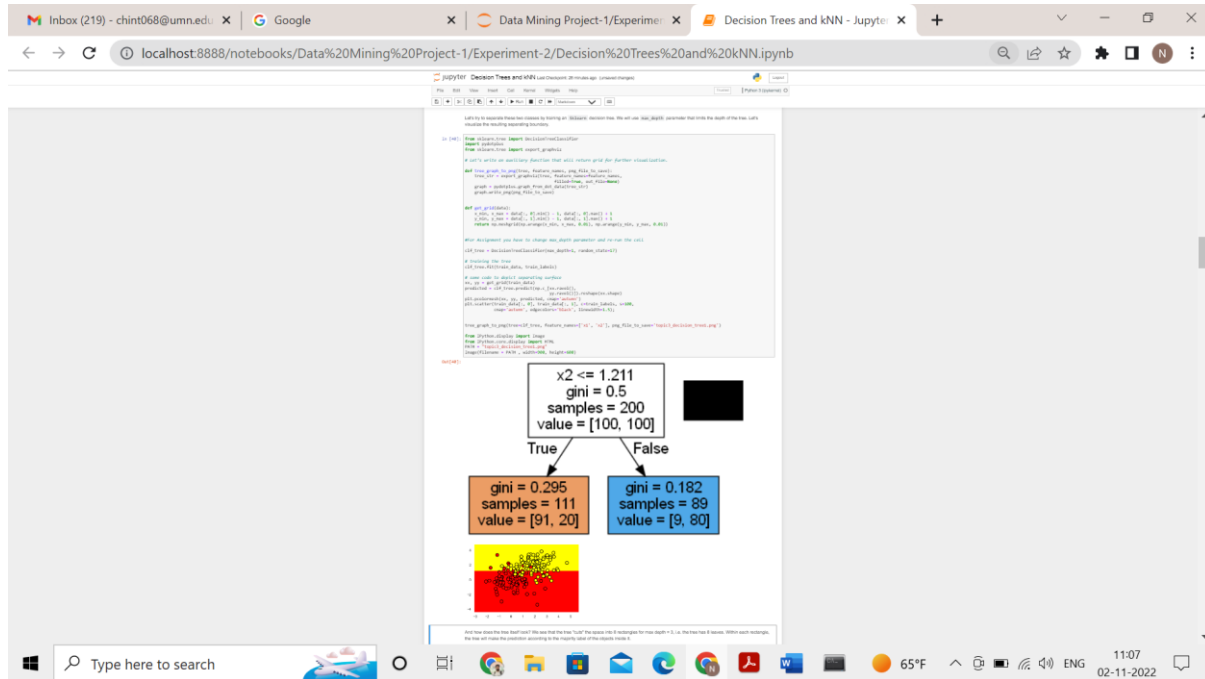


For Overfit (max_depth=9)

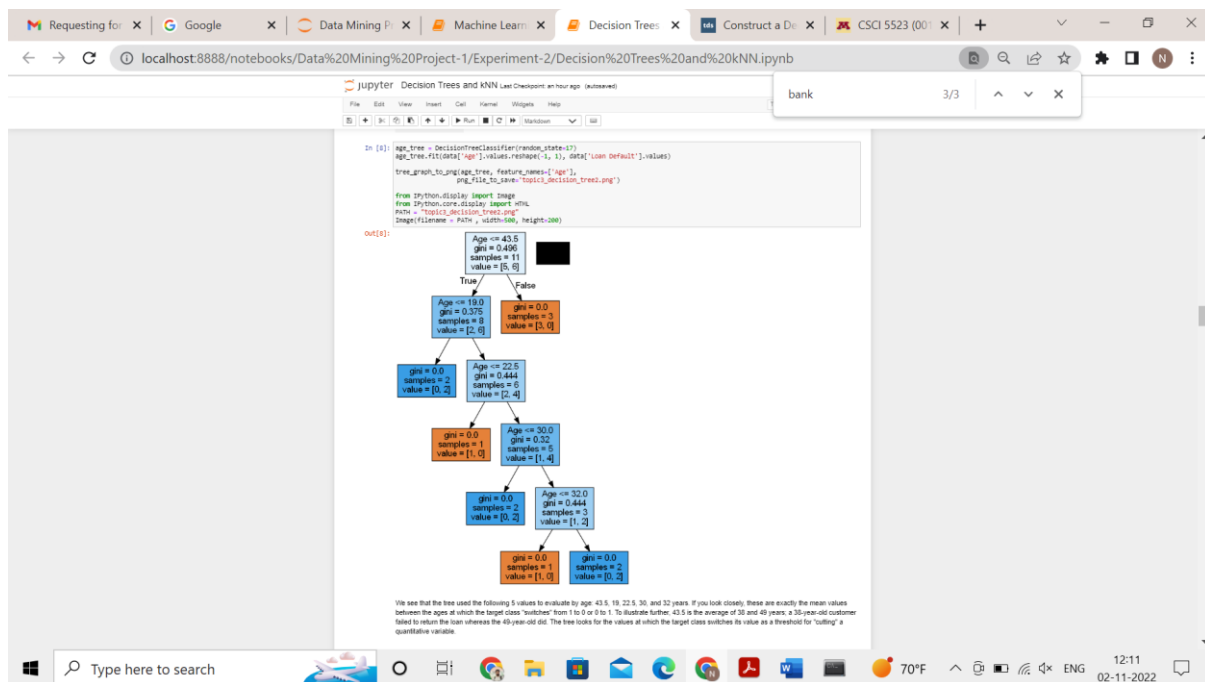


For Underfit (max_depth=1)

Project - 1



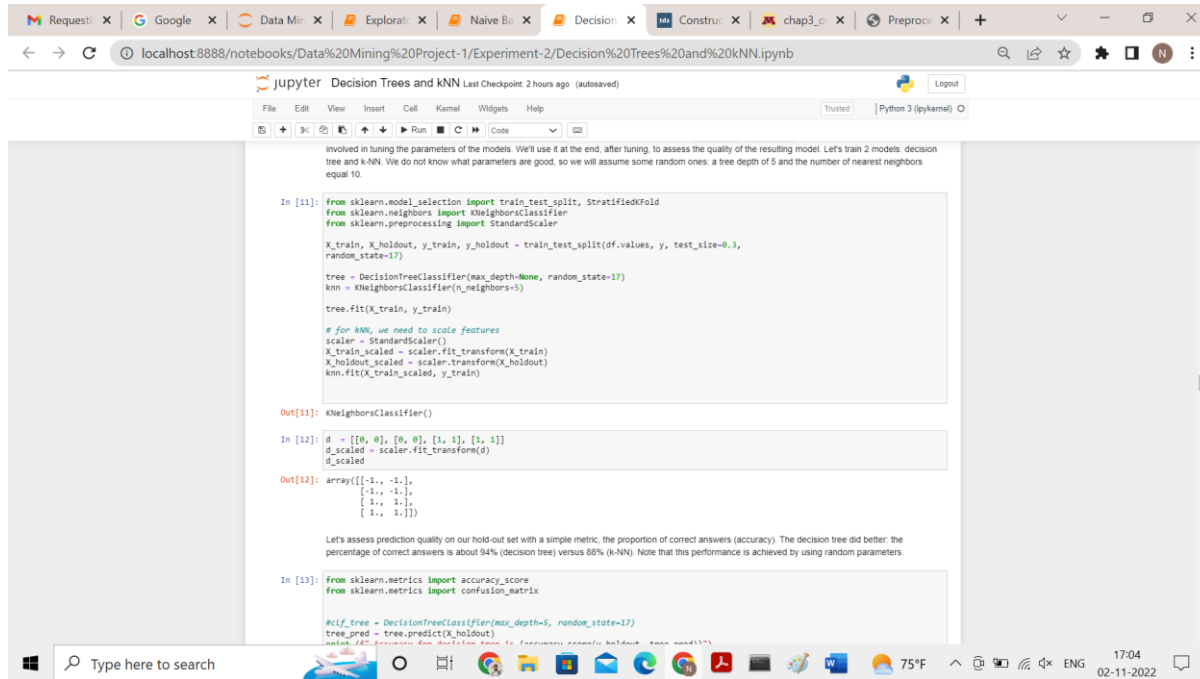
3.



We see that the tree used the following 5 values to evaluate by age: 43.5, 19, 22.5, 30, and 32 years. If you look closely, these are exactly the mean values between the ages at which the target class "switches" from 1 to 0 or 0 to 1.

4. Standard scalar standardizes the features by removing the mean and scaling to unit variance.

Project - 1



The screenshot shows a Jupyter Notebook interface with the following content:

involved in tuning the parameters of the models. We'll use it at the end, after tuning, to assess the quality of the resulting model. Let's train 2 models: decision tree and k-NN. We do not know what parameters are good, so we will assume some random ones: a tree depth of 5 and the number of nearest neighbors equal 10.

```
In [11]: from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

X_train, X_holdout, y_train, y_holdout = train_test_split(df.values, y, test_size=0.3,
random_state=17)

tree = DecisionTreeClassifier(max_depth=None, random_state=17)
knn = KNeighborsClassifier(n_neighbors=5)

tree.fit(X_train, y_train)

# for KNN, we need to scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_holdout_scaled = scaler.transform(X_holdout)
knn.fit(X_train_scaled, y_train)
```

```
Out[11]: KNeighborsClassifier()
```

```
In [12]: d = [[0, 0], [0, 0], [1, 1], [1, 1]]
d_scaled = scaler.fit_transform(d)
d_scaled
```

```
Out[12]: array([[[-1., -1.],
[-1., -1.],
[ 1.,  1.],
[ 1.,  1.]])
```

Let's assess prediction quality on our hold-out set with a simple metric: the proportion of correct answers (accuracy). The decision tree did better: the percentage of correct answers is about 94% (decision tree) versus 88% (k-NN). Note that this performance is achieved by using random parameters.

```
In [13]: from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

#clf_tree = DecisionTreeClassifier(max_depth=5, random_state=17)
tree_pred = tree.predict(X_holdout)
#confusion_matrix(tree_pred, y_holdout) # Parameters: sklearn.metrics.confusion_matrix(y_true, y_pred)
```

The scalar transformations are ([[-1., -1.], [-1., -1.], [1., 1.], [1., 1.]])

5. Small decision tree generally leads to under-fitting and big decision tree generally leads to overfitting. The training and test accuracy for overfitting decision tree model and under fitted decision tree model is performed against small and big data. Overfit model has high accuracy for training data but poor accuracy for the test data; in overfit model, test data accuracy is slightly better in the big data than small data because it has relatively high variance and low bias. Underfit model has poor accuracy for both training and test data irrespective of small data or big data because it high bias and low variance.

Project - 1

6.

```

=====
Balanced and imbalanced dataset statistics
=====
For Balance Training Dataset, shape is (700, 18) and we have 350 positive class, and 350 negative class
For Imbalance Training Dataset, shape is (1993, 18) and we have 10 positive class, and 1983 negative class
For Balance Testing Dataset, shape is (200, 18) and we have 133 positive class, and 133 negative class
For Imbalance Testing Dataset, shape is (872, 18) and we have 5 positive class, and 867 negative class

=====
Train on balance and test on imbalance
=====
[[350  0]
 [ 0 350]]
=====
Train Accuracy for decision tree trained on balanced data is 1.0
[[0.88 1.79]
 [ 0  5]]
=====
while test accuracy on imbalanced data is 0.7947247786422018
Train [f1_score, precision, recall] for decision tree trained on balanced data is (1.0, 1.0, 1.0)
while [f1_score, precision, recall] on imbalanced data is (0.05291005291005291, 0.02717391304347826, 1.0)

=====
Train on imbalance and test on balance
=====
[[350  0]
 [ 0 350]]
=====
Train Accuracy for decision tree trained on balanced data is 1.0
[[1.00 25]
 [30 103]]
=====
while test accuracy on balanced data is 0.793233882706767
Train [f1_score, precision, recall] for decision tree trained on balanced data is (1.0, 1.0, 1.0)
while [f1_score, precision, recall] on balanced data is (0.7892720306513409, 0.8046875, 0.7744360902255639)

=====
Train on imbalance and test on imbalance
=====
[[1983  0]
 [  0 10]]
=====
Train Accuracy for decision tree trained on imbalanced data is 1.0
[[133  0]
 [112 21]]
=====
while test accuracy on balanced data is 0.5789473684210527
Train [f1_score, precision, recall] for decision tree trained on imbalanced data is (1.0, 1.0, 1.0)
while [f1_score, precision, recall] on imbalanced data is (0.2727272727272727, 1.0, 0.15789473684210525)

=====
Train on imbalance and test on imbalance
=====
[[1983  0]
 [  0 10]]
=====
Train Accuracy for decision tree trained on imbalanced data is 1.0
[[864  3]
 [ 5  0]]
=====
while test accuracy on imbalanced data is 0.9988256888733948
Train [f1_score, precision, recall] for decision tree trained on imbalanced data is (1.0, 1.0, 1.0)
while [f1_score, precision, recall] on imbalanced data is (0.0, 0.0, 0.0)

```

First the statistics of the balanced and imbalanced data is seen. The model is trained on balanced dataset and tested on imbalanced dataset and same process is performed on other combinations too such as training on balanced data set and testing on balanced dataset; training on imbalanced dataset and testing on balanced dataset; training and testing on imbalanced dataset.

70% of the set is allocated to training and 30% for holdout. With this split the model is trained and tested and a confusion matrix is created for each case and accordingly the corresponding F1 score, precision and recall are calculated using the in built sklearn functions.

	F1 score	Precision	Recall
Training balanced	1.0	1.0	1.0
Testing imbalanced	0.05291005291005291	0.02717391304347826	1.0
Training balanced	1.0	1.0	1.0
Testing balanced	0.7892720306513409	0.8046875	0.7744360902255639
Training imbalanced	1.0	1.0	1.0
Testing balanced	0.2727272727272727	1.0	0.15789473684210525
Training imbalanced	1.0	1.0	1.0

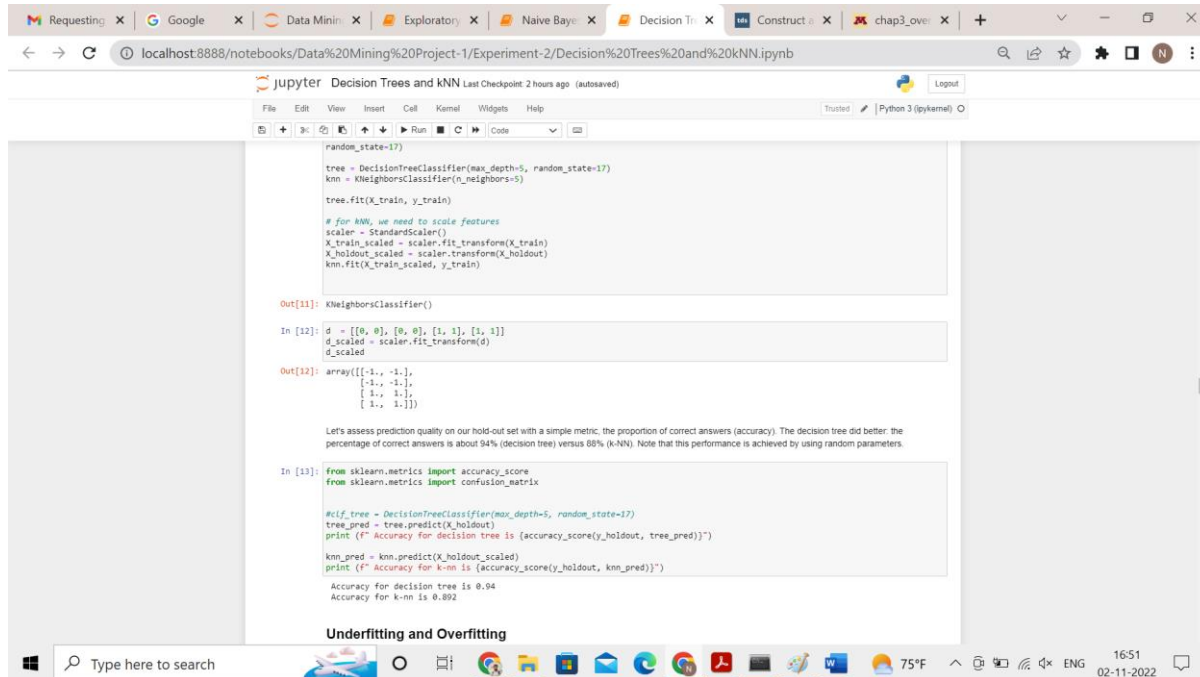
Project - 1

Testing imbalanced	0.0	0.0	0.0
--------------------	-----	-----	-----

7.

In the experiment, the first the decision tree model is applied to the dataset and then some noise is added (irrelevant attributes) to both the test and training set and executed to check for the results. Adding irrelevant attributes reduces the accuracy of the model over the test dataset because of the model overfitting.

8.



```
random_state=17)
tree = DecisionTreeClassifier(max_depth=5, random_state=17)
knn = KNeighborsClassifier(n_neighbors=5)
tree.fit(X_train, y_train)

# for kNN, we need to scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_holdout_scaled = scaler.transform(X_holdout)
knn.fit(X_train_scaled, y_train)

Out[11]: KNeighborsClassifier()

In [12]: d = [[0, 0], [0, 0], [1, 1], [1, 1]]
d_scaled = scaler.fit_transform(d)
d_scaled

Out[12]: array([[ 1., -1.],
               [-1., -1.],
               [ 1.,  1.],
               [ 1.,  1.]])

Let's assess prediction quality on our hold-out set with a simple metric: the proportion of correct answers (accuracy). The decision tree did better: the percentage of correct answers is about 94% (decision tree) versus 88% (k-NN). Note that this performance is achieved by using random parameters.

In [13]: from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

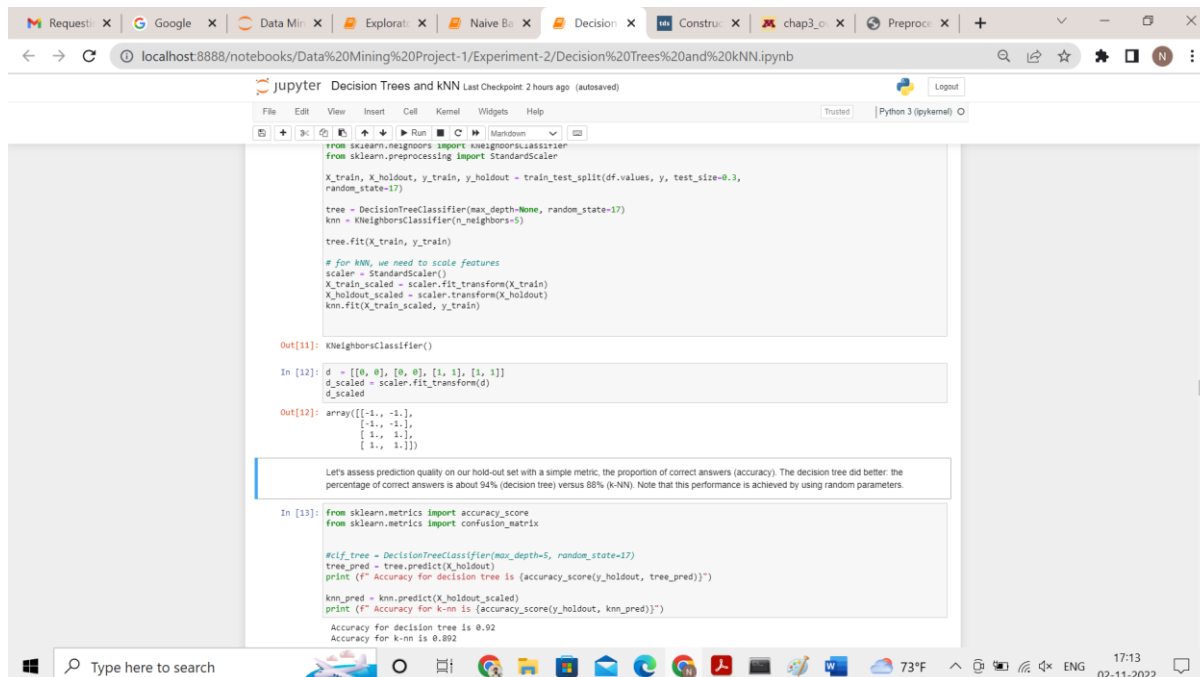
# If tree = DecisionTreeClassifier(max_depth=5, random_state=17)
tree_pred = tree.predict(X_holdout)
print(f"Accuracy for decision tree is {accuracy_score(y_holdout, tree_pred)}")

knn_pred = knn.predict(X_holdout_scaled)
print(f"Accuracy for k-nn is {accuracy_score(y_holdout, knn_pred)}")

Accuracy for decision tree is 0.94
Accuracy for k-nn is 0.882
```

Underfitting and Overfitting

For max depth =5, the accuracy is 94%



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

X_train, X_holdout, y_train, y_holdout = train_test_split(df.values, y, test_size=0.3,
random_state=17)

tree = DecisionTreeClassifier(max_depth=None, random_state=17)
knn = KNeighborsClassifier(n_neighbors=5)
tree.fit(X_train, y_train)

# for kNN, we need to scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_holdout_scaled = scaler.transform(X_holdout)
knn.fit(X_train_scaled, y_train)

Out[11]: KNeighborsClassifier()

In [12]: d = [[0, 0], [0, 0], [1, 1], [1, 1]]
d_scaled = scaler.fit_transform(d)
d_scaled

Out[12]: array([[ 1., -1.],
               [-1., -1.],
               [ 1.,  1.],
               [ 1.,  1.]])

Let's assess prediction quality on our hold-out set with a simple metric: the proportion of correct answers (accuracy). The decision tree did better: the percentage of correct answers is about 94% (decision tree) versus 88% (k-NN). Note that this performance is achieved by using random parameters.

In [13]: from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

# If tree = DecisionTreeClassifier(max_depth=5, random_state=17)
tree_pred = tree.predict(X_holdout)
print(f"Accuracy for decision tree is {accuracy_score(y_holdout, tree_pred)}")

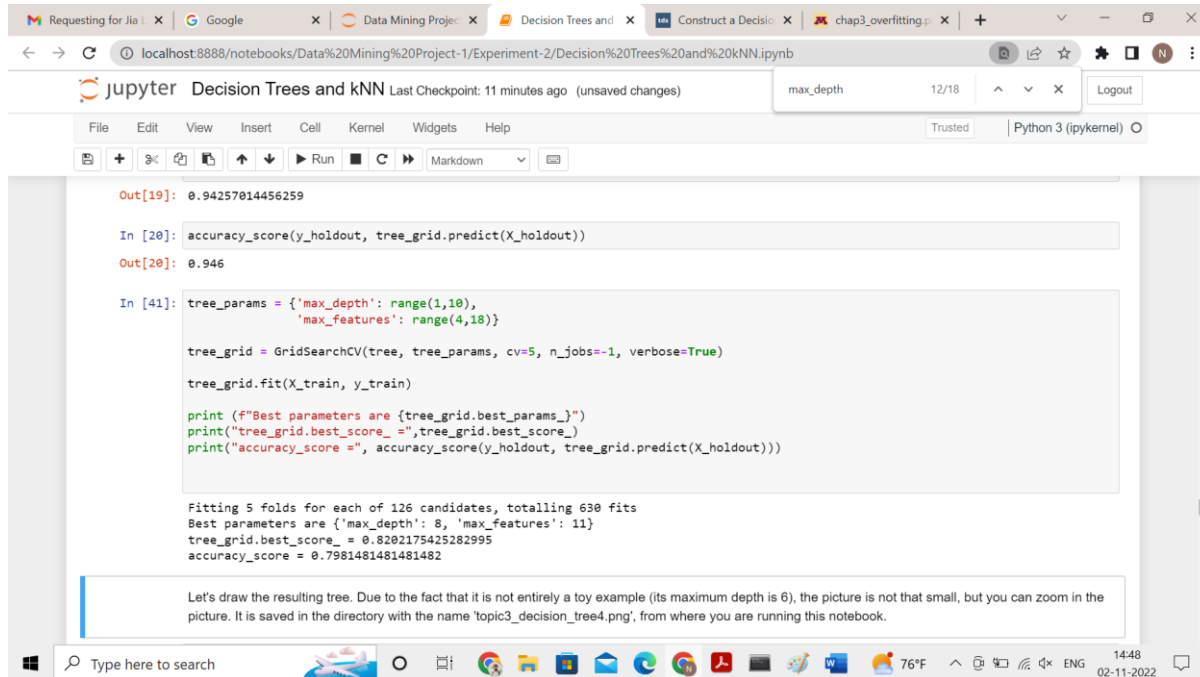
knn_pred = knn.predict(X_holdout_scaled)
print(f"Accuracy for k-nn is {accuracy_score(y_holdout, knn_pred)}")

Accuracy for decision tree is 0.92
Accuracy for k-nn is 0.882
```

For max depth is default, the accuracy is 92%. There is a decrease in the accuracy because of the overfitting of the model when max_depth is default value, when max_depth is default, the tree expands till the leaf nodes reach their pure state.

Project - 1

9.



```
Out[19]: 0.94257814456259

In [20]: accuracy_score(y_holdout, tree_grid.predict(X_holdout))

Out[20]: 0.946

In [41]: tree_params = {'max_depth': range(1,10),
                        'max_features': range(4,18)}

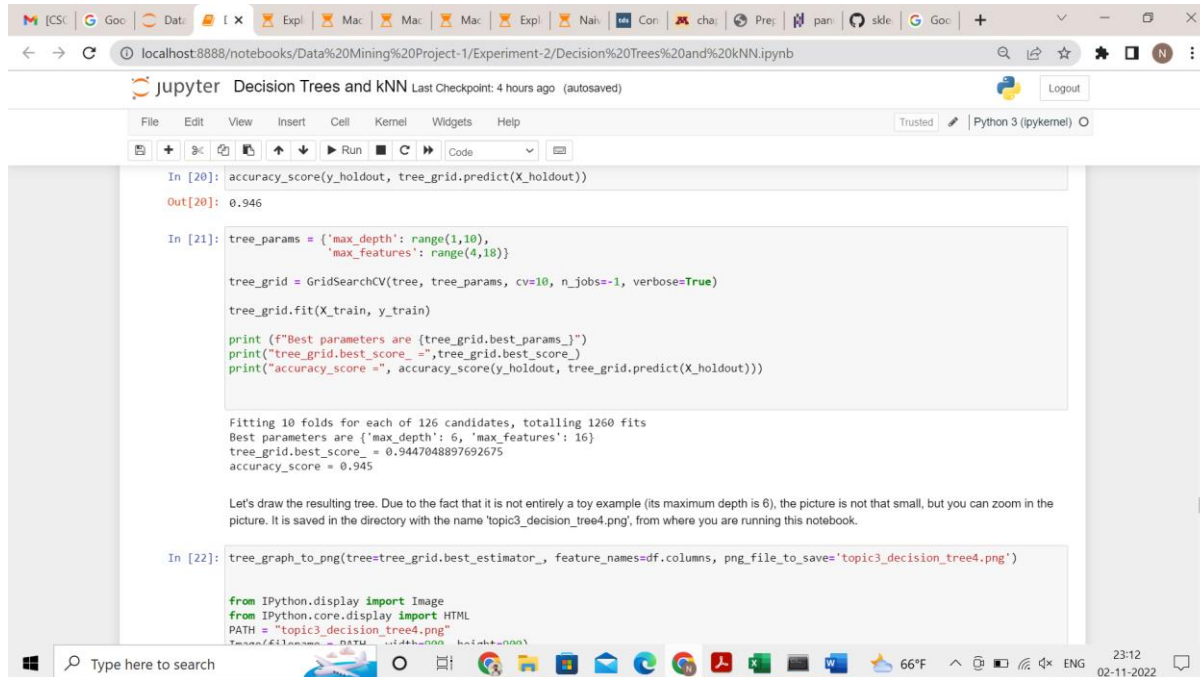
tree_grid = GridSearchCV(tree, tree_params, cv=5, n_jobs=-1, verbose=True)
tree_grid.fit(X_train, y_train)

print(f"Best parameters are {tree_grid.best_params_}")
print("tree_grid.best_score_ =", tree_grid.best_score_)
print("accuracy_score =", accuracy_score(y_holdout, tree_grid.predict(X_holdout)))

Fitting 5 folds for each of 126 candidates, totalling 630 fits
Best parameters are {'max_depth': 8, 'max_features': 11}
tree_grid.best_score_ = 0.8202175425282995
accuracy_score = 0.7981481481481482

Let's draw the resulting tree. Due to the fact that it is not entirely a toy example (its maximum depth is 6), the picture is not that small, but you can zoom in the picture. It is saved in the directory with the name 'topic3_decision_tree4.png', from where you are running this notebook.
```

A total of 630 decision trees can be constructed



```
In [20]: accuracy_score(y_holdout, tree_grid.predict(X_holdout))

Out[20]: 0.946

In [21]: tree_params = {'max_depth': range(1,10),
                        'max_features': range(4,18)}

tree_grid = GridSearchCV(tree, tree_params, cv=10, n_jobs=-1, verbose=True)
tree_grid.fit(X_train, y_train)

print(f"Best parameters are {tree_grid.best_params_}")
print("tree_grid.best_score_ =", tree_grid.best_score_)
print("accuracy_score =", accuracy_score(y_holdout, tree_grid.predict(X_holdout)))

Fitting 10 folds for each of 126 candidates, totalling 1260 fits
Best parameters are {'max_depth': 6, 'max_features': 16}
tree_grid.best_score_ = 0.9447048997692675
accuracy_score = 0.945

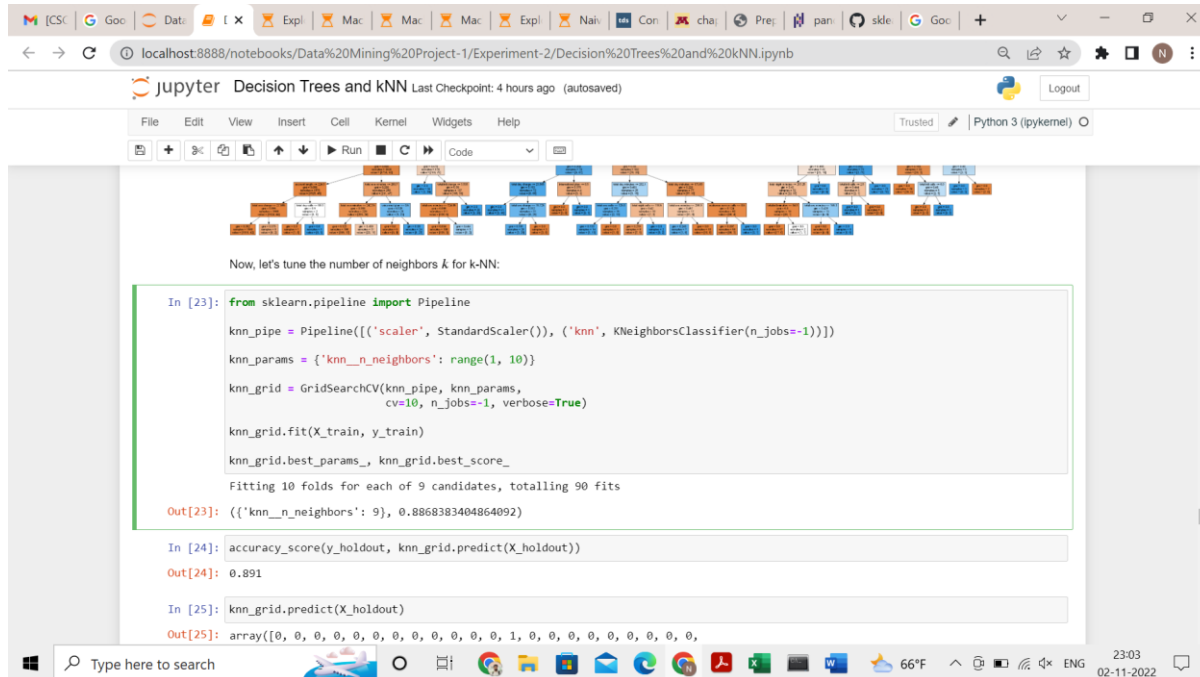
Let's draw the resulting tree. Due to the fact that it is not entirely a toy example (its maximum depth is 6), the picture is not that small, but you can zoom in the picture. It is saved in the directory with the name 'topic3_decision_tree4.png', from where you are running this notebook.

In [22]: tree_graph_to_png(tree=tree_grid.best_estimator_, feature_names=df.columns, png_file_to_save='topic3_decision_tree4.png')

from IPython.display import Image
from IPython.core.display import HTML
PATH = "topic3_decision_tree4.png"
Image(PATH)
```

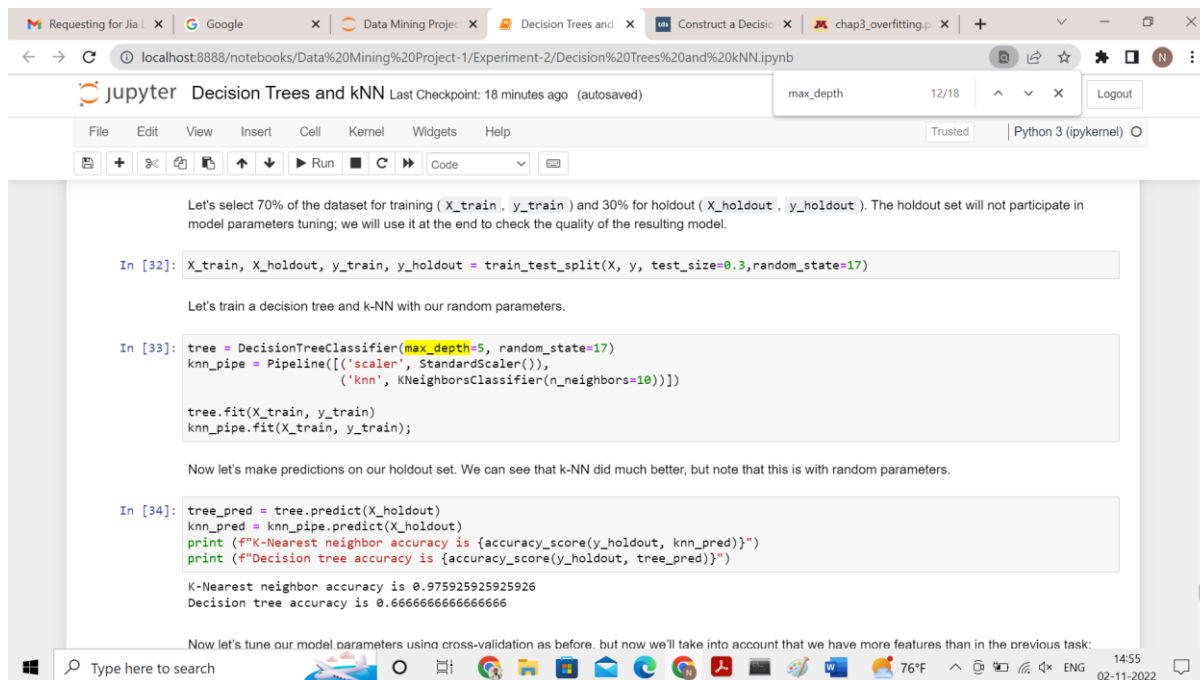
A total of 1260 decision trees can be constructed.

10.



k=9 is the best choice

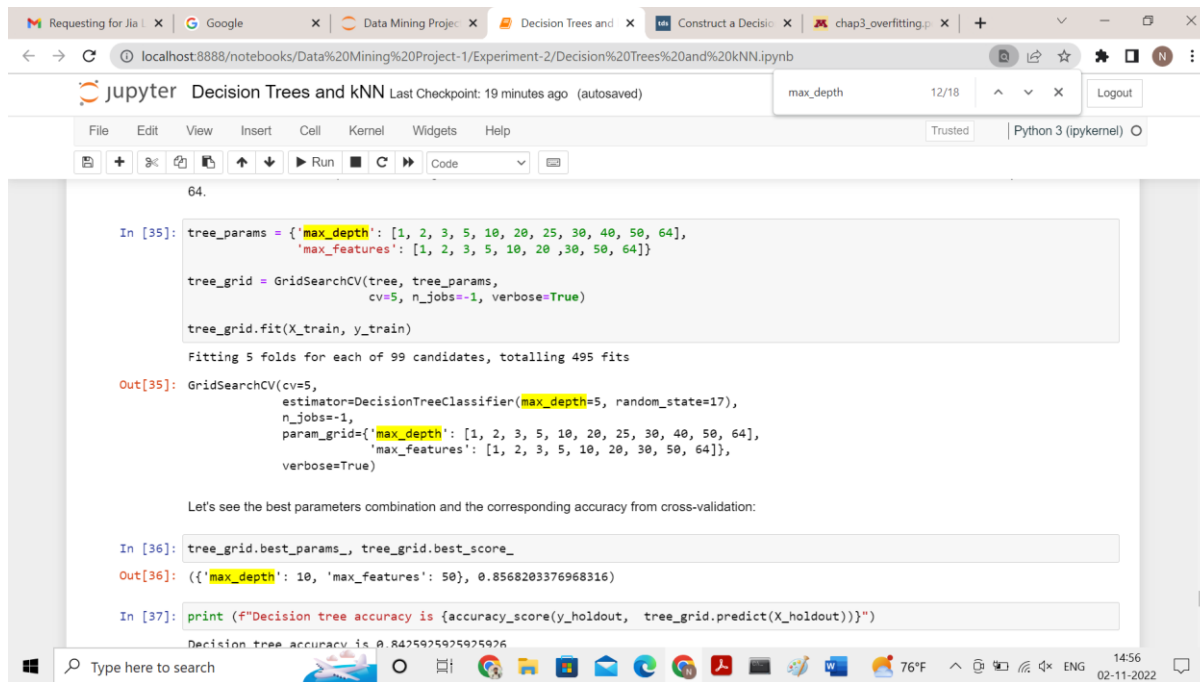
11.



Accuracy of the decision tree for max depth = 5 is 66.67%

Accuracy of the kNN when K=10 is 97.59%

Project - 1



```
64.

In [35]: tree_params = {'max_depth': [1, 2, 3, 5, 10, 20, 25, 30, 40, 50, 64],
                        'max_features': [1, 2, 3, 5, 10, 20, 30, 50, 64]}

tree_grid = GridSearchCV(tree, tree_params,
                        cv=5, n_jobs=-1, verbose=True)

tree_grid.fit(X_train, y_train)

Fitting 5 folds for each of 99 candidates, totalling 495 fits

Out[35]: GridSearchCV(cv=5,
                    estimator=DecisionTreeClassifier(max_depth=5, random_state=17),
                    n_jobs=-1,
                    param_grid={'max_depth': [1, 2, 3, 5, 10, 20, 25, 30, 40, 50, 64],
                                'max_features': [1, 2, 3, 5, 10, 20, 30, 50, 64]},
                    verbose=True)

Let's see the best parameters combination and the corresponding accuracy from cross-validation:

In [36]: tree_grid.best_params_, tree_grid.best_score_

Out[36]: ({'max_depth': 10, 'max_features': 50}, 0.8568203376968316)

In [37]: print (f"Decision tree accuracy is {accuracy_score(y_holdout, tree_grid.predict(X_holdout))}")

Decision tree accuracy is 0.8475925925925926
```

With 5-fold cross validation, best parameters are `max_depth = 10` and `max_features=50` and accuracy of holdout dataset for the decision tree = 85.68%