CM1

February 25, 2021

1 [CM1] Seeds dataset (Preprocessing and Algorithms)

1.1 Data Pre-processing

1.1.1 Importing Libraries

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import tree
from sklearn.model_selection import KFold,GridSearchCV,train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings("ignore")
```

1.1.2 Loading dataset

[2]:
```
seeds_data = pd.read_csv('seeds_dataset.txt', sep="\t", error_bad_lines=False,␣
 ↪warn_bad_lines=False)
seeds_data.
 ↪columns=['area','perimeter','compactness','length_kernel','width_kernel','asymmetry_coeff',
```

Here, dataset is skipping the lines with extra tabs.

1.1.3 First 5 columns of dataset

[3]:
```
seeds_data.head()
```

[3]:

	area	perimeter	compactness	length_kernel	width_kernel	\
0	14.88	14.57	0.8811	5.554	3.333	
1	14.29	14.09	0.9050	5.291	3.337	
2	13.84	13.94	0.8955	5.324	3.379	
3	16.14	14.99	0.9034	5.658	3.562	
4	14.38	14.21	0.8951	5.386	3.312	

	asymmetry_coeff	length_of_kernel_groove	target
0	1.018	4.956	1
1	2.699	4.825	1
2	2.259	4.805	1
3	1.355	5.175	1
4	2.462	4.956	1

1.1.4 Overview of Dataset

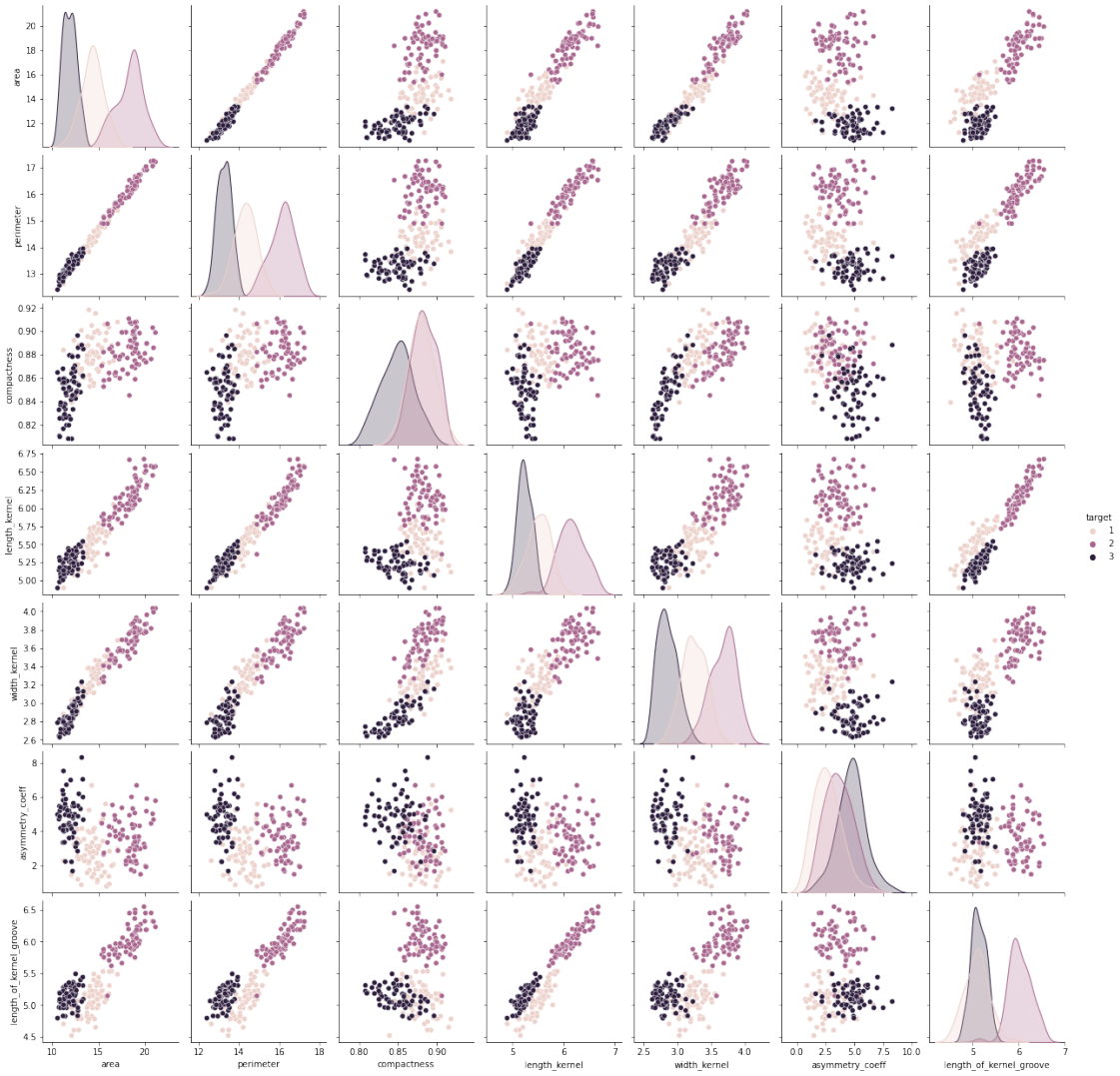
[4]:
```
seeds_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 198 entries, 0 to 197
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   area                    198 non-null    float64
 1   perimeter               198 non-null    float64
 2   compactness             198 non-null    float64
 3   length_kernel           198 non-null    float64
 4   width_kernel            198 non-null    float64
 5   asymmetry_coeff         198 non-null    float64
 6   length_of_kernel_groove 198 non-null    float64
 7   target                  198 non-null    int64
dtypes: float64(7), int64(1)
memory usage: 12.5 KB
```

1.1.5 Plotting features

[5]:
```
sns.pairplot(seeds_data, hue="target")
```

[5]:
```
<seaborn.axisgrid.PairGrid at 0x2b9ebebbc10>
```



In above plots, we can see that all 3 different varieties of wheat: Kama, Rosa and Canadian are easily separable using any features.

1.1.6 Dividing data into train-test sets

[6]:
```
X = seeds_data.iloc[:,0:7].values
y = seeds_data.iloc[:,7].values

# Without Standardization
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)

# With Standardization
scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

We have divided 20% dataset for testing and 80% for training and validation.

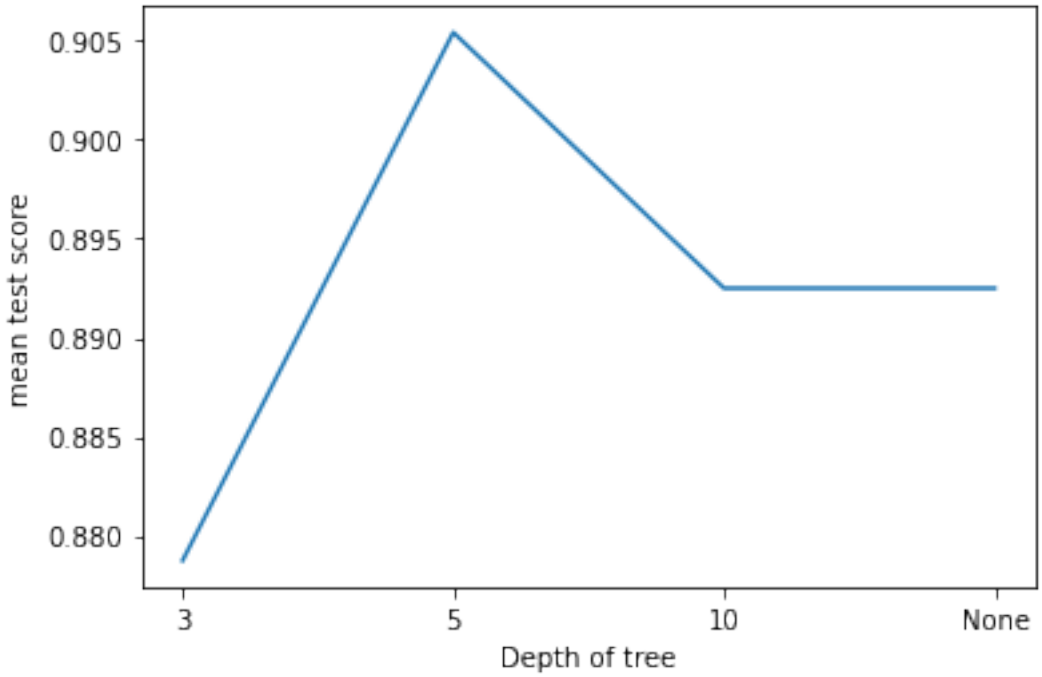
1.2 Algorithm 1 : Decision Tree Algorithm (Without Standardization)

1.2.1 Applying algorithm on training set

[7]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'max_depth':[3, 5, 10, None]}

classifier = GridSearchCV(DecisionTreeClassifier(random_state= 0),␣
↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
classifier = classifier.fit(X_train, y_train)

print(classifier.best_params_)
results = classifier.cv_results_
print(results['mean_test_score'])

max_depth=[3, 5, 10, None]
max_depth1 = list(map(str,max_depth))
plt.plot(max_depth1, results['mean_test_score'])
plt.xlabel("Depth of tree")
plt.ylabel("mean test score")
```

```
{'max_depth': 5}
[0.87875    0.90541667 0.8925     0.8925    ]
```

[7]:
```
Text(0, 0.5, 'mean test score')
```



1.2.2 Applying algorithm on test set

[8]:
```
clf = classifier.best_estimator_
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
print(accuracy_score(y_test, predictions))
```

```
0.9
```

1.3 Algorithm 1 : Decision Tree Algorithm (With Standardization)

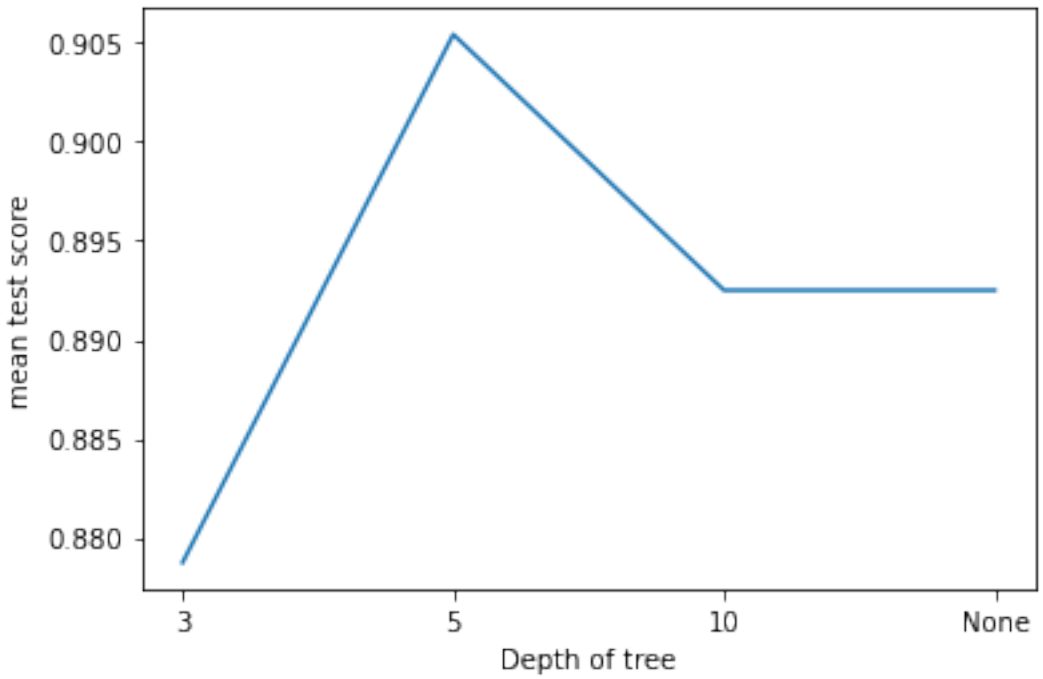
1.3.1 Applying algorithm on training set

[9]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid = {'max_depth':[3, 5, 10, None]}

classifier = GridSearchCV(DecisionTreeClassifier(random_state= 0),␣
 ↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
classifier = classifier.fit(X_train1, y_train1)

print(classifier.best_params_)
results = classifier.cv_results_
print(results['mean_test_score'])

max_depth=[3, 5, 10, None]
```

```
max_depth1 = list(map(str,max_depth))
plt.plot(max_depth1, results['mean_test_score'])
plt.xlabel("Depth of tree")
plt.ylabel("mean test score")
```

```
{'max_depth': 5}
[0.87875    0.90541667 0.8925     0.8925    ]
```

[9]: `Text(0, 0.5, 'mean test score')`



1.3.2 Applying algorithm on test set

[10]:
```
clf = classifier.best_estimator_
clf.fit(X_train1, y_train1)
predictions = clf.predict(X_test1)
print(accuracy_score(y_test1, predictions))
```

```
0.9
```

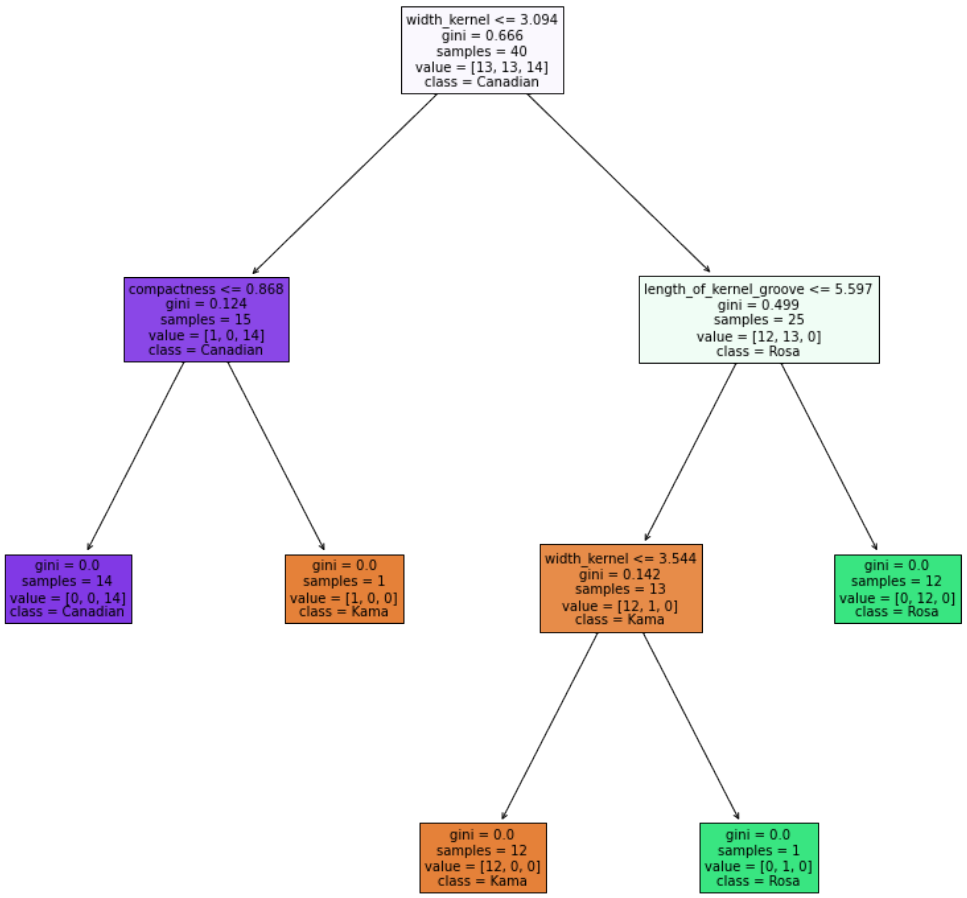
1.3.3 Observation

As we see in the plot, On train set we got highest accuracy 90.5, and best parameter given by GridSearchCV is {'max_depth': 5}. With this best parameter, we got 90 accuracy on test set.

Also we can see that, there is no difference in accuracy with or without standardization.

1.3.4 Tree with splitting rules

[11]:
```
decision_tree = DecisionTreeClassifier(random_state=0, max_depth=5)
decision_tree = decision_tree.fit(X_test, y_test)
fig = plt.figure(figsize=(15,15))
plt.show(tree.
 ↪plot_tree(decision_tree,fontsize=10,filled=True,class_names=['Kama','Rosa','Canadian'],feat
```



The tree first splits on feature 'width_kernel' value <= 3.094, and attempts to split the 'canadian' seeds from others. So, the left hand side child is left with 'Canadian' seeds and 1 'Kama' seed, which are further divided into leaves on the basis of 'compactness' feature.
Right hand side child is still left with 2 classes : 'Kama' and 'Rosa', which are split on the feature 'length_of_kernel_groove', and we get 'Rosa' seeds with length_of_kernel_groove > 5.597.
Again, 'width_kernel' feature is used to differentiate 1 'Rosa' seed from 'Kama's seeds. As a result,

each leaf is left with single class of target.

Thus, Data points can be divided into target classes, based on the features : 'width_kernel','compactness' and 'length_of_kernel_groove'.

1.4 Algorithm 2 : Random Forest Algorithm (Without Standardization)

1.4.1 Applying algorithm on training set

[12]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid={'n_estimators': [5, 10, 50, 150, 200], 'max_depth' :[3, 5, 10,␣
 ↪None]}

classifier = GridSearchCV(RandomForestClassifier(random_state= 0),␣
 ↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
classifier = classifier.fit(X_train, y_train)

print(classifier.best_params_)
results = classifier.cv_results_
print(results['mean_test_score'])
```

```
{'max_depth': 5, 'n_estimators': 10}
[0.91041667 0.91083333 0.91083333 0.89833333 0.89833333 0.89791667
 0.92375    0.92375    0.91125    0.91083333 0.88541667 0.92375
 0.91125    0.9175     0.91083333 0.88541667 0.92375    0.91125
 0.9175     0.91083333]
```

1.4.2 Applying algorithm on test set

[13]:
```
clf = classifier.best_estimator_
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
print(accuracy_score(y_test, predictions))
```

```
0.9
```

1.5 Algorithm 2 : Random Forest Algorithm (With Standardization)

1.5.1 Applying algorithm on training set

[14]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid={'n_estimators': [5, 10, 50, 150, 200], 'max_depth' :[3, 5, 10,␣
 ↪None]}

classifier = GridSearchCV(RandomForestClassifier(random_state= 0),␣
 ↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
classifier = classifier.fit(X_train1, y_train1)

print(classifier.best_params_)
```

```
results = classifier.cv_results_
print(results['mean_test_score'])
```

```
{'max_depth': 5, 'n_estimators': 10}
[0.91041667 0.91083333 0.91083333 0.89833333 0.89833333 0.89791667
 0.92375    0.92375    0.91125    0.91083333 0.88541667 0.92375
 0.91125    0.9175     0.91083333 0.88541667 0.92375    0.91125
 0.9175     0.91083333]
```

1.5.2 Applying algorithm on test set

[15]:
```
clf = classifier.best_estimator_
clf.fit(X_train1, y_train1)
predictions = clf.predict(X_test1)
print(accuracy_score(y_test1, predictions))
```

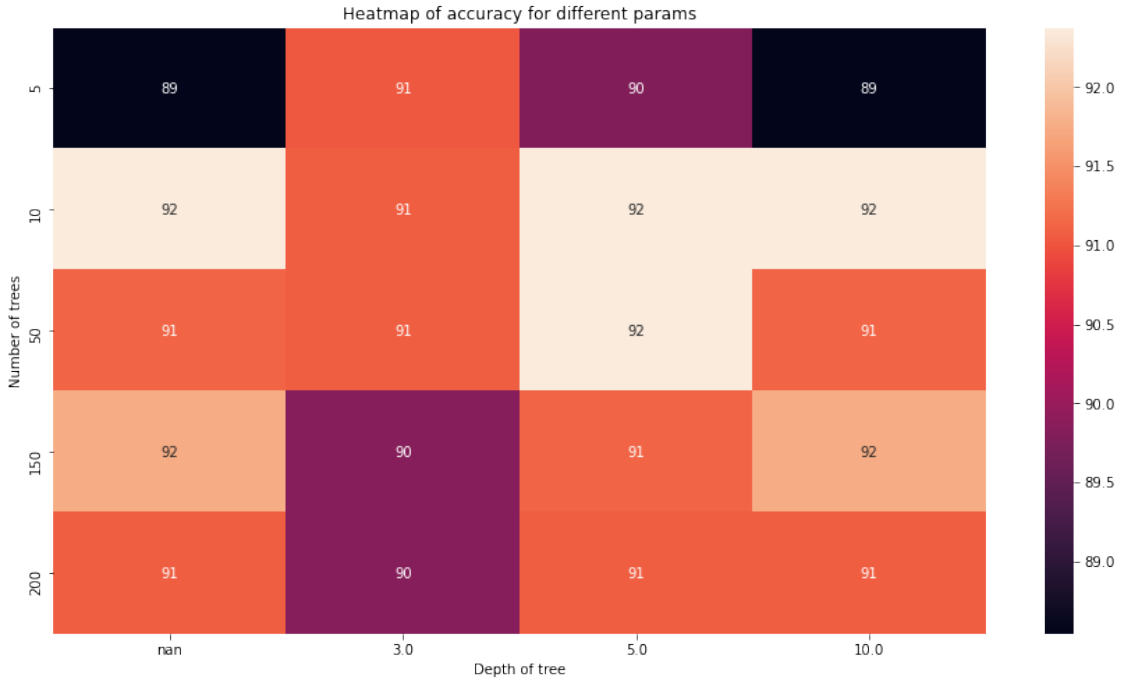
```
0.9
```

1.5.3 Observation

On train set we got highest accuracy 92.37, and best parameter given by GridSearchCV is {'max_depth': 5, 'n_estimators': 10}. With this best parameter, we got 90 accuracy on test set.
Also we can see that, there is no difference in accuracy with or without standardization.

1.5.4 Plotting heatmap on best params

[16]:
```
accuracy_df = pd.DataFrame(results['params'])
accuracy_df["accuracy"] = results['mean_test_score']*100
accuracy_df = accuracy_df.
 ↪pivot(columns='max_depth',index='n_estimators',values='accuracy')

plt.figure(figsize=(15,8))
sns.heatmap(data=accuracy_df, annot=True)
plt.title("Heatmap of accuracy for different params")
plt.xlabel("Depth of tree")
plt.ylabel("Number of trees")
plt.show()
```



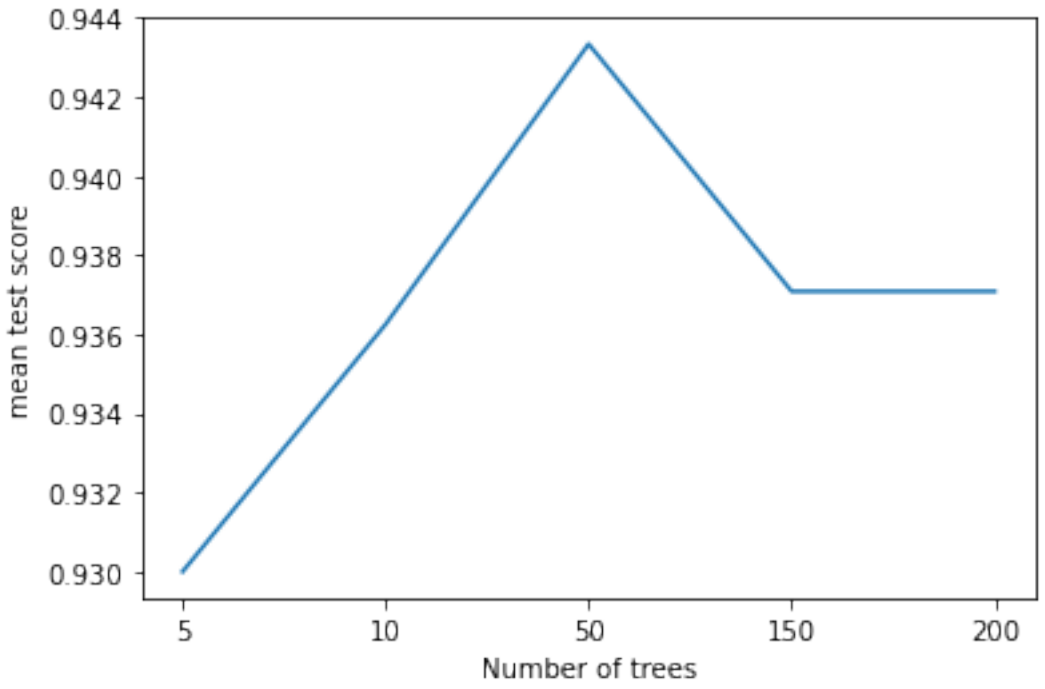
1.6 Algorithm 3 : Gradient Tree Boosting Algorithm (Without Standardization)

1.6.1 Applying algorithm on training set

[17]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid={'n_estimators': [5, 10, 50, 150, 200]}

classifier = GridSearchCV(GradientBoostingClassifier(random_state= 0),␣
 ↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
classifier = classifier.fit(X_train, y_train)

print(classifier.best_params_)
results = classifier.cv_results_
print(results['mean_test_score'])

n_estimators = [5, 10, 50, 150, 200]
n_estimators1 = list(map(str,n_estimators))
plt.plot(n_estimators1, results['mean_test_score'])
plt.xlabel("Number of trees")
plt.ylabel("mean test score")
```

```
{'n_estimators': 50}
[0.93       0.93625    0.94333333 0.93708333 0.93708333]
```

[17]: Text(0, 0.5, 'mean test score')



1.6.2 Applying algorithm on test set

[18]:
```
clf = classifier.best_estimator_
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
print(accuracy_score(y_test, predictions))
```

0.975

1.7 Algorithm 3 : Gradient Tree Boosting Algorithm (With Standardization)

1.7.1 Applying algorithm on training set

[ ]:
```
kf = KFold(random_state=0,n_splits=10)
param_grid={'n_estimators': [5, 10, 50, 150, 200]}

classifier = GridSearchCV(GradientBoostingClassifier(random_state= 0),␣
 ↪param_grid=param_grid, scoring='accuracy', cv=kf, n_jobs=-1)
classifier = classifier.fit(X_train1, y_train1)

print(classifier.best_params_)
results = classifier.cv_results_
print(results['mean_test_score'])
```

```
n_estimators = [5, 10, 50, 150, 200]
n_estimators1 = list(map(str,n_estimators))
plt.plot(n_estimators1, results['mean_test_score'])
plt.xlabel("Number of trees")
plt.ylabel("mean test score")
```

1.7.2 Applying algorithm on test set

[ ]:
```
clf = classifier.best_estimator_
clf.fit(X_train1, y_train1)
predictions = clf.predict(X_test1)
print(accuracy_score(y_test1, predictions))
```

1.7.3 Observation

As we see in the plot, On train set we got highest accuracy 94.33, and best parameter given by GridSearchCV is {'n_estimators': 50}. With this best parameter, we got 97.5 accuracy on test set. Also we can see that, there is no difference in test accuracy with or without standardization.

1.8 References

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html