Abstract

This report is in fullfilment for phase 2 of the 2 phase machine learning project demonstrating an end to end machine learning project. In phase 1 of the project, we loaded, preprocessed and explored the data. The data was exported to file.

In this phase, we continue from there on by loading the prepared data & performing data modeling. As part of modeling we take steps for spliting the data in train test pair, feature selection and hyper parameter tuning with k-fold cross validation. We later use models with tuned parameters to measure their performance on testig data. We use four different multiclass classifiers to perform modeling activities which are, KNN, Decision Tree, Random Forest & Gaussian Naive Bayes. In the end we make comparisions using statistical tests to identify best performing models. Some important the aspects of modeling revolved around avoiding overfitting and tuning of parameters for the models and visualizing the effect of change in the model performance with varying values of parameters.

Introduction

Objective of phase 2 of ML Project is to carry out a detailed Data Modeling using different Classifiers and comparing the performances of these classifiers with each other. As our problem is multiclass classification, we will use the Accuracy of the model to evaluate the performance.

To complete the data modeling part of the project, we will start off with loading the prepared data from two files, descriptive_data.csv & target_data.csv. We then used Min Max scaler to scale the data before preforming further modeling tasks. We then proceed towards hyper parameter tuning. In this section we try to find optimal values for the classifier specific parameters. We use 5 fold cross validation to compare the performance of the models with different parameter with each other. We perform tuning phase with 4 models, KNN, Decision Tree, Random Forest & Naive Bayes classifiers. Occupying the feature selection technique to get particular number of significant features, we tried to find out the best features for that particular model. Later on, we used the tuned models to measure the actual performance on testing data and compared the models based on their accuracy.

Preliminaries

Firstly we will load required packages to support machine learning classification implementation. We will be using pandas, numpy, seaborn, matplotlib, IPython, sklearn, scipy and mpl_toolkits libraries.

In phase one, we created and stored data of preprocessed descriptive and target data seperately in 'CSV' files. We will load data from these files in 'descriptives' and 'target' dataframes respectively. Original dataset is hosted on "UCI Machine Learning Library" and is put together by "Jock A. Blackard", "Dr. Denis J. Dean" & "Dr. Charles W. Anderson". We will apply basic data preparation steps like rename columns, describe data and check the head of dataframe to ensure proper data loading.

In [1]:
```
# To display output of all the statements and not only last statement
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import warnings
warnings.simplefilter("ignore")

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
from IPython.display import display, HTML
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.preprocessing import MinMaxScaler
from pandas.plotting import parallel_coordinates
from sklearn.metrics import precision_recall_fscore_support as score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, GridSearchCV

from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from scipy import stats
from sklearn import metrics

from mpl_toolkits.mplot3d import Axes3D
```

In [2]:
```
descriptives_file = "descriptive_data.csv"
target_file = "target_data.csv"
```

In [3]:
```
data_features = ['elevation', 'aspect', 'slope', 'horizontal_distance_to_hydro
logy',
            'vertical_distance_to_hydrology', 'horizontal_distance_to_roadway
s',
            'hillshade_9am', 'hillshade_noon', 'hillshade_3pm', \
            'horizontal_distance_to_fire_points',
            'rawah','neota','comanche_peak','cache_la_poudre',
            '2702', '2703', '2704', '2705', '2706', '2717', '3501', \
            '3502', '4201', '4703', '4704', '4744', '4758', '5101', \
            '5151', '6101', '6102', '6731', '7101', '7102', '7103', \
            '7201', '7202', '7700', '7701', '7702', '7709', '7710', \
            '7745', '7746', '7755', '7756','7757', '7790', '8703', \
            '8707', '8708', '8771', '8772', '8776']

label_feature = ['cover_type']

# data is encoded , undersampled, preprocessed
descriptives = pd.read_csv(descriptives_file, names = data_features)
target = pd.read_csv(target_file, names = label_feature)

print("Descriptive Features Check:")
print("Shape:")
descriptives.shape
descriptives.head()

print(" ")
print("Target Feature Check:")
print("Shape:")
target.shape
target.head()
```

```
Descriptive Features Check:
Shape:
```

Out[3]: (19229, 54)

Out[3]:

	elevation	aspect	slope	horizontal_distance_to_hydrology	vertical_distance_to_hydrology	hor
0	3067	5	15	108	16	
1	3152	32	10	660	135	
2	2977	345	5	120	6	
3	3221	106	9	192	1	
4	3194	4	15	499	67	

5 rows × 54 columns

```
Target Feature Check:
Shape:
```

Out[3]: (19229, 1)

Out[3]:

	cover_type
0	1
1	1
2	1
3	1
4	1

Scaling

Most of the machine learning algorithm works well with scaled data. Highly varying magnitude of data impacts outcome of algorithm which uses distance based techniques. Therefore it is always preferable to scale data before feeding it to alorithms. We will use MinMaxScaler to scale values between 0 to 1. Cosider that it preserves the orginal distribution of data and doesn't reduce importance of outliers.

In [4]:
```
# Scaling
scaler_min = MinMaxScaler()
descriptive_minmax = scaler_min.fit_transform(descriptives)
```

```
C:\Users\Vishwa\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:33
4: DataConversionWarning: Data with input dtype int64 were all converted to f
loat64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

Feature selection

We have 54 descriptive features including continuos and encoded features. In this section, we will identify 10 relevent features contributing more to target feature cover type on the bases of calculated feature importance scores. Feature selection will be included into pipeline later on. This analysis plays signinficant role to improve model efficiency and avoid overfitting.

RandomForestClassifier generated importance scores and list of 10 relevent features in the section below. Also, we can derive from the ranking plot for score that features like elevation, horizontal distances has higher relevence than hillshade features. During pipelining we will explore the subset of features which works better with perticular model.

In [5]:
```
# Feature Selection
print("Feature Selection using RandomForestClassifier:")
features_count = 20
model_rfi_selection = RandomForestClassifier(n_estimators=100)
model_rfi_selection.fit(descriptives, target)
fs_indices_rfi = np.argsort(model_rfi_selection.feature_importances_)[::-1][0:
features_count]

best_features_rfi_selection = descriptives.columns[fs_indices_rfi].values
print("----------------------------------------------------------------------------
--------")
print("Best Selected features:")
best_features_rfi_selection

feature_importances_rfi = model_rfi_selection.feature_importances_[fs_indices_
rfi]
print("----------------------------------------------------------------------------
--------")
print("Feature Importance scores generated for best selected features:")
feature_importances_rfi
```

```
C:\Users\Vishwa\Anaconda3\lib\site-packages\ipykernel_launcher.py:6: DataConv
ersionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples,), for example using ravel().


Feature Selection using RandomForestClassifier:
```

Out[5]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
----------------------------------------------------------------------------
--
Best Selected features:
```

Out[5]:
```
array(['elevation', 'horizontal_distance_to_roadways',
       'horizontal_distance_to_fire_points',
       'horizontal_distance_to_hydrology', 'hillshade_9am',
       'vertical_distance_to_hydrology', 'aspect', 'hillshade_3pm',
       'cache_la_poudre', 'hillshade_noon', 'slope', '4703', '8771',
       '8772', 'comanche_peak', '2704', 'rawah', '2705', '8776', '7746'],
      dtype=object)
```

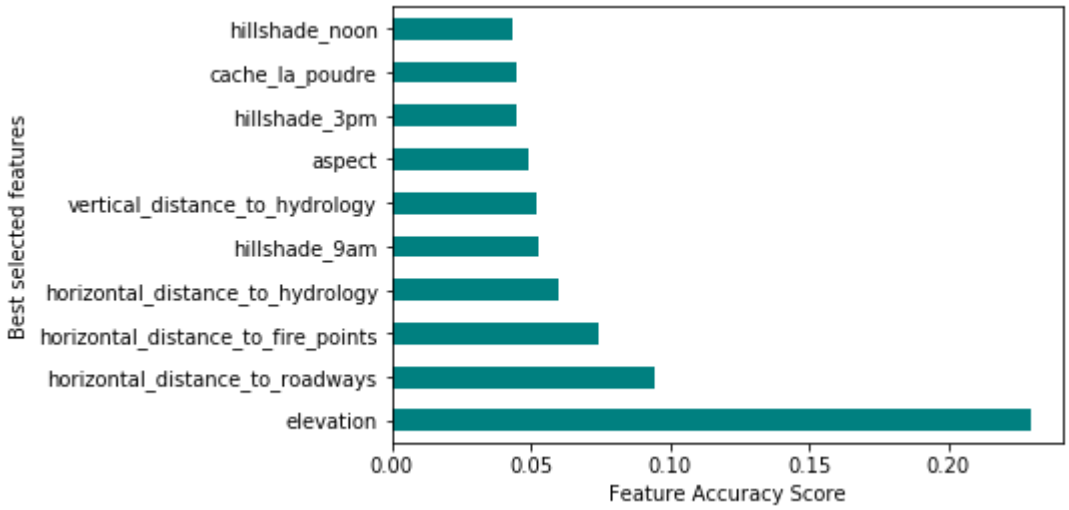
```
----------------------------------------------------------------------------
--
Feature Importance scores generated for best selected features:
```

Out[5]:
```
array([0.22921592, 0.09474663, 0.07440739, 0.06018066, 0.05239024,
       0.05219365, 0.0489309 , 0.04460849, 0.04454453, 0.04347245,
       0.03737224, 0.02435812, 0.01891863, 0.01823384, 0.01781939,
       0.01656767, 0.01428166, 0.01387598, 0.00872232, 0.00820142])
```

Let's visualize the result using features importance to get overview of target feature dependency.

In [6]:
```
print("Visual representation of selected features")
# Plot graph for feature selection
feature_importances = pd.Series(feature_importances_rfi, index=best_features_rfi_selection)
feature_importances.nlargest(10).plot(kind='barh', color= 'teal')
plt.xlabel("Feature Accuracy Score")
plt.ylabel("Best selected features")
plt.show();
```

```
Visual representation of selected features
```



Data Sampling

Original dataset has 581012 observations with uneven distribution of target feature 'cover type'. This highly imbalanced nature of data set can adversely affect the performance of most of the machine learning algorithms. Cover type has only 2747 instances of ‘‘Cottonwood/Willow’ category which is the minimum count among target categories. To overcome this issue we have already undersampled the dateset using RandomUnderSampler. As we have sufficient number of instances in minority category, we have chosen undersampling technique to resample the data. Final undersampled data to be used for model fitting has 19229 observations.

Data Split

Here, we divide total 19229 observations into training and testing sets for descriptive features and target features in split of 80:20 of train:test data. We will use train_test_split() with 0.20 test_size parameter value to perform this operation.

In [7]:
```
# Data split
x_train, x_test, y_train, y_test = train_test_split(descriptive_minmax, target
, test_size = 0.20, random_state = 0)

print("x_train Shape")
x_train.shape
print("x_test Shape")
x_test.shape
```

```
x_train Shape
```

Out[7]:
```
(15383, 54)
```

```
x_test Shape
```

Out[7]:
```
(3846, 54)
```

Model Evaluation Strategy

We have chosen to use accuracy as evaluation metric, as our problem is multiclass classification problem and accuracy is prefered choice for these type of problems.

We will evaluate our models with paired t-test on accuracy. We will use 5-fold crossvalidation with pipeline to perform gird search to find optimal parameter values for hyper parameter tuning. We will use around 16000 instances for cross validation and tuning, and later on will use around 4000 instances to test our models with optimal parameters.

We have multiple functions which will help us perform feature selection & extracting results from the pipeline. Function are provided by RMIT University Course tutorial - https://rmit.instructure.com/courses/50786/files/7870346?fd_cookie_set=1 (https://rmit.instructure.com/courses/50786/files/7870346?fd_cookie_set=1).

In [8]:
```
# Hyper parameter tuning
cv_method = StratifiedKFold(n_splits=5, random_state=999)
```

In [9]:
```
# custom function for feature selection using RFI inside a pipeline
# n_estimators=100 is used
class FeatureSelectorRFI(BaseEstimator, TransformerMixin):

    # class constructor
    # class attributes must end with a "_"
    def __init__(self, n_features_=10):
        self.n_features_ = n_features_
        self.fs_indices_ = None

    # override the fit function
    def fit(self, X, y):
        from sklearn.ensemble import RandomForestClassifier
        from numpy import argsort
        model_rfi = RandomForestClassifier(n_estimators=100)
        model_rfi.fit(X, y)
        self.fs_indices_ = argsort(model_rfi.feature_importances_)[::-1][0:sel
f.n_features_]
        return self

    # override the transform function
    def transform(self, X, y=None):
        return X[:, self.fs_indices_]
```

In [10]:
```
# custom function to format the search results as a Pandas data frame

def get_search_results(gs):

    def model_result(scores, params):
        scores = {'mean_score': np.mean(scores),
             'std_score': np.std(scores),
             'min_score': np.min(scores),
             'max_score': np.max(scores)}
        return pd.Series({**params,**scores})

    models = []
    scores = []

    for i in range(gs.n_splits_):
        key = f"split{i}_test_score"
        r = gs.cv_results_[key]
        scores.append(r.reshape(-1,1))

    all_scores = np.hstack(scores)
    for p, s in zip(gs.cv_results_['params'], all_scores):
        models.append((model_result(s, p)))

    pipe_results = pd.concat(models, axis=1).T.sort_values(['mean_score'], asc
ending=False)

    columns_first = ['mean_score', 'std_score', 'max_score', 'min_score']
    columns = columns_first + [c for c in pipe_results.columns if c not in col
umns_first]

    return pipe_results[columns]
```

Hyper parameter tuning

We will use pipeline to perform gridsearch on training dataset to find optimal value of parameteres for different classification algorithms. To compare the performance, we will use K-Fold Cross Validation technique to calculate the effective accuracy of models.

We will use the optimal parameters found by this grid search to run all the models on testing dataset to measure the actual performance of the models.

KNN Classifier

To tune the parameters of KNN algorithm, we will try out two parameters, Number of neighbors & Distance metric (Manhattan or Euclidean). We will use 10, 20, 30, 40 & 54 highest ranked features, along with 1 to 10 neighbors for both p = 1 & p = 2 distances.

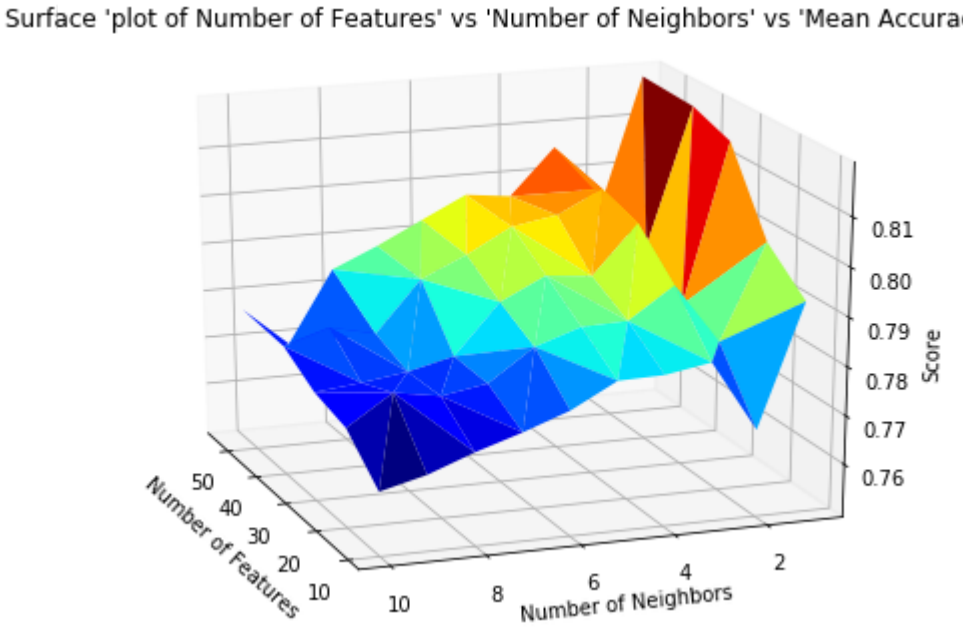
In [11]:
```
#KNN

pipe_KNN = Pipeline(steps=[('rfi_fs', FeatureSelectorRFI()),
                           ('knn', KNeighborsClassifier())])

params_pipe_KNN = {'rfi_fs__n_features_': [10, 20, 30, 40, descriptives.shape[
1]],
                   'knn__n_neighbors': [1,2,3,4,5,6,7,8,9,10],
                   'knn__p': [1, 2]}

gs_pipe_KNN = GridSearchCV(estimator=pipe_KNN,
                           param_grid=params_pipe_KNN,
                           cv=cv_method,
                           refit=True,
                           n_jobs=-2,
                           scoring='accuracy',
                           verbose=1)


gs_pipe_KNN.fit(x_train, y_train);
```

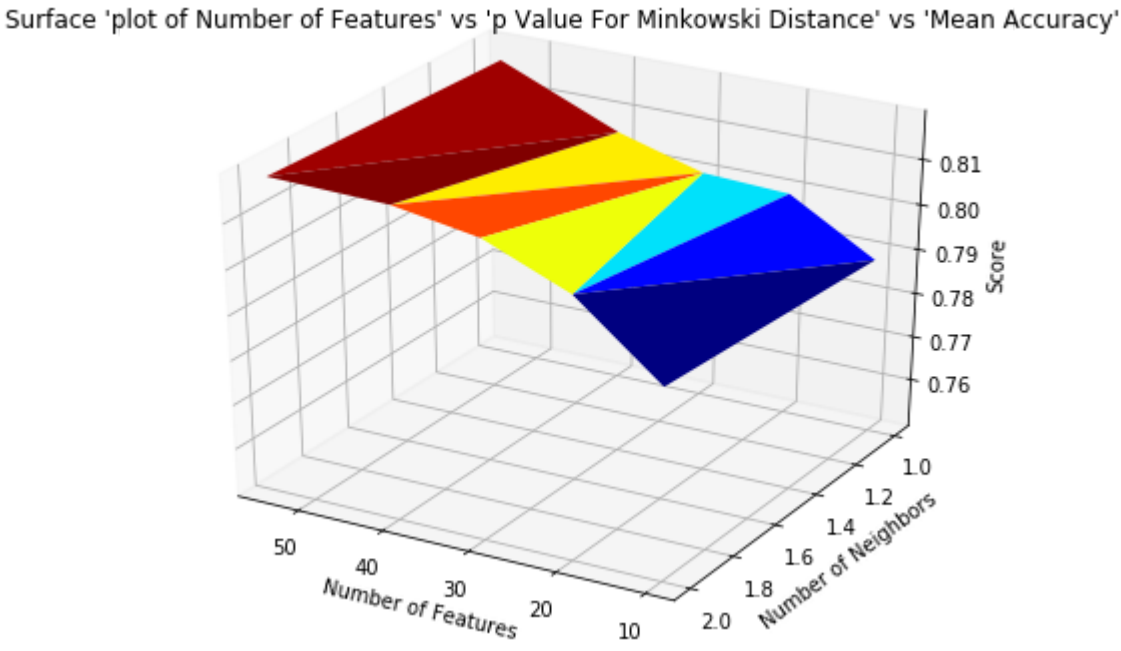
```
Fitting 5 folds for each of 100 candidates, totalling 500 fits

[Parallel(n_jobs=-2)]: Using backend LokyBackend with 7 concurrent workers.
[Parallel(n_jobs=-2)]: Done  36 tasks      | elapsed:  1.9min
[Parallel(n_jobs=-2)]: Done 186 tasks      | elapsed: 12.2min
[Parallel(n_jobs=-2)]: Done 436 tasks      | elapsed: 28.4min
[Parallel(n_jobs=-2)]: Done 500 out of 500 | elapsed: 31.5min finished
C:\Users\Vishwa\Anaconda3\lib\site-packages\ipykernel_launcher.py:16: DataCon
versionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples,), for example using ravel().
  app.launch_new_instance()
C:\Users\Vishwa\Anaconda3\lib\site-packages\sklearn\pipeline.py:267: DataConv
ersionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples, ), for example using ravel().
  self._final_estimator.fit(Xt, y, **fit_params)
```

Analysis - Grid Search KNN

From the 3d surface plots below, we can see how the change in parameter value affects the accuracy. It turns out that models using all 54 features to train have the highest performance. Euclidean distance performs slightly better than manhattan distance. However, p = 2 is extreme condition and we should check for the further values of p, but due to significant processing time and relatively low increase in the preformance of the models for p = 1 & p = 2, we will not evaluate for more values of p.

In [12]:
```
results_KNN = get_search_results(gs_pipe_KNN)

fig = plt.figure(figsize = (8,6))
ax = fig.gca(projection='3d')
ax.plot_trisurf(results_KNN['rfi_fs__n_features_'], results_KNN['knn__n_neighb
ors'], results_KNN['mean_score'], cmap=plt.cm.jet, linewidth=0.2)
_ = ax.set_xlabel("Number of Features")
_ = ax.set_ylabel("Number of Neighbors")
_ = ax.set_zlabel("Score")
_ = ax.set_title("Surface 'plot of Number of Features' vs 'Number of Neighbor
s' vs 'Mean Accuracy' ")
_ = ax.view_init(20, 160)
plt.show();

fig = plt.figure(figsize = (8,6))
ax = fig.gca(projection='3d')
ax.plot_trisurf(results_KNN['rfi_fs__n_features_'], results_KNN['knn__p'], res
ults_KNN['mean_score'], cmap=plt.cm.jet, linewidth=0.2)
_ = ax.set_xlabel("Number of Features")
_ = ax.set_ylabel("Number of Neighbors")
_ = ax.set_zlabel("Score")
_ = ax.set_title("Surface 'plot of Number of Features' vs 'p Value For Minkows
ki Distance' vs 'Mean Accuracy' ")
_ = ax.view_init(30, 120)
plt.show();
```





Results

Optimal values for parameters are

- Number of features = 54,
- P (Distance metric) = 2,
- Number of Neighbors = 1

Highest Accuracy = 81.9281 %

In [13]:
```
gs_pipe_KNN.best_params_
```

Out[13]:
```
{'knn__n_neighbors': 1, 'knn__p': 2, 'rfi_fs__n_features_': 54}
```

In [14]:
```
gs_pipe_KNN.best_score_
```

Out[14]:
```
0.8192810245075733
```

Decision Tree Classifier

We will search for the optimal values for maximum depth and minimum sample size for split. We will check with different number of features. The ranges used for search are as follows

- Number of features - 10, 20, 30, 40, 54
- Max Depth - 5, 10, 15, 20, 25, 30, 35
- Min samples for split - 2, 5, 7

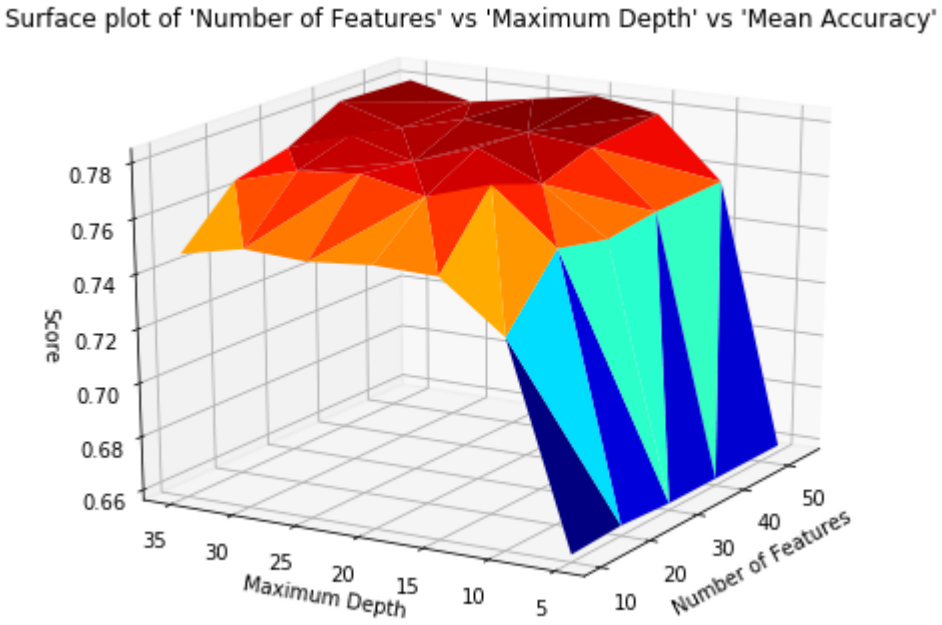
In [15]:
```
# Decision Tree

pipe_DT = Pipeline([('rfi_fs', FeatureSelectorRFI()),
                    ('dt', DecisionTreeClassifier(criterion='gini'))])

params_pipe_DT = {'rfi_fs__n_features_': [10, 20, 30, 40, descriptives.shape[1
]],
                  'dt__max_depth': [ 5, 10, 15, 20, 25, 30, 35],
                  'dt__min_samples_split': [2, 5, 7],
                 }

gs_pipe_DT = GridSearchCV(estimator=pipe_DT,
                          param_grid=params_pipe_DT,
                          cv=cv_method,
                          refit=True,
                          n_jobs=-2,
                          scoring='accuracy',
                          verbose=1)

gs_pipe_DT.fit(x_train, y_train);
```

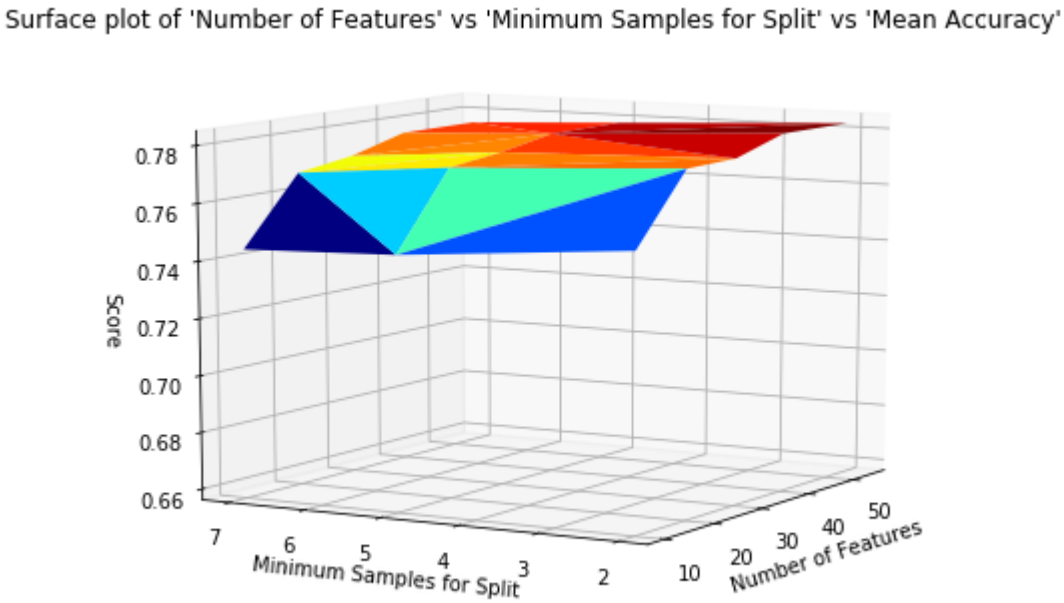
```
Fitting 5 folds for each of 105 candidates, totalling 525 fits

[Parallel(n_jobs=-2)]: Using backend LokyBackend with 7 concurrent workers.
[Parallel(n_jobs=-2)]: Done  36 tasks      | elapsed:   25.4s
[Parallel(n_jobs=-2)]: Done 186 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-2)]: Done 436 tasks      | elapsed:  5.0min
[Parallel(n_jobs=-2)]: Done 525 out of 525 | elapsed:  6.0min finished
C:\Users\Vishwa\Anaconda3\lib\site-packages\ipykernel_launcher.py:16: DataCon
versionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples,), for example using ravel().
  app.launch_new_instance()
```

Analysis - Grid Search Decision Tree

It can be seen from the 3d Surface plots below that the performance of models improves with increaseing values for number of features and maximum depth. We get best performance near 40 features and 30 Max depth. However, lower values for min_samples_split perfroms better than higher values.

In [16]:
```
gs_pipe_DT_sr = get_search_results(gs_pipe_DT)

fig = plt.figure(figsize = (8,6))
ax = fig.gca(projection='3d')
ax.plot_trisurf(gs_pipe_DT_sr['rfi_fs__n_features_'], gs_pipe_DT_sr['dt__max_d
epth'], gs_pipe_DT_sr['mean_score'], cmap=plt.cm.jet, linewidth=0.2)
_ = ax.set_xlabel("Number of Features")
_ = ax.set_ylabel("Maximum Depth")
_ = ax.set_zlabel("Score")
_ = ax.set_title("Surface plot of 'Number of Features' vs 'Maximum Depth' vs
 'Mean Accuracy' ")
_ = ax.view_init(20, 210)
plt.show();

fig = plt.figure(figsize = (8,6))
ax = fig.gca(projection='3d')
ax.plot_trisurf(gs_pipe_DT_sr['rfi_fs__n_features_'], gs_pipe_DT_sr['dt__min_s
amples_split'], gs_pipe_DT_sr['mean_score'], cmap=plt.cm.jet, linewidth=0.2)
_ = ax.set_xlabel("Number of Features")
_ = ax.set_ylabel("Minimum Samples for Split")
_ = ax.set_zlabel("Score")
_ = ax.set_title("Surface plot of 'Number of Features' vs 'Minimum Samples for
Split' vs 'Mean Accuracy' ")
_ = ax.view_init(10, 210)
plt.show();
```





Results

Optimal values for parameters are

- Number of features = 54,
- Maximum Depth = 30,
- Min Samples for split = 40

Highest Accuracy = 78.326 %

In [17]:
```
gs_pipe_DT.best_params_
```

Out[17]:
```
{'dt__max_depth': 30, 'dt__min_samples_split': 2, 'rfi_fs__n_features_': 40}
```

In [18]:
```
gs_pipe_DT.best_score_
```

Out[18]:
```
0.782487161151921
```

Random Forest Classifier

Grid search for random forest classifier is quite exhastive process in terms of computation time, due to repeatative sub-sampling.

- Number of features - 10, 20, 30, 40, 54
- Max Depth - 10, 20, 30, 40
- Min samples for split - 2, 5, 7
- Min Samples at leaf node - 1, 3, 10
- Method to calculate IG = 'Gini' & 'Entropy'

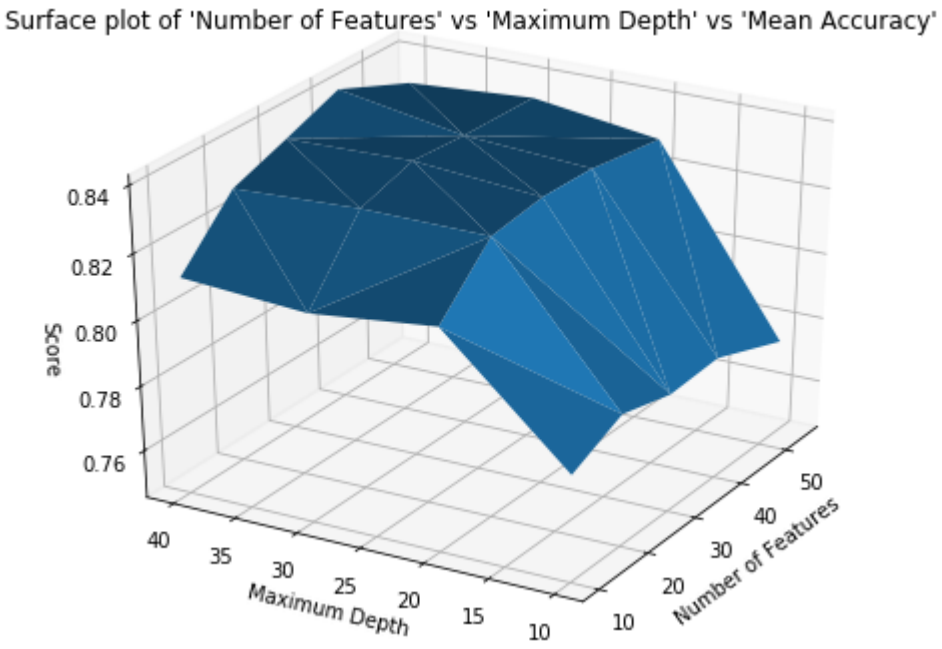
In [19]:
```
# Random Forest
pipe_RF = Pipeline([('rfi_fs', FeatureSelectorRFI()),
                    ('rf', RandomForestClassifier())])

params_pipe_RF = {'rfi_fs__n_features_': [10, 20,30,40, descriptives.shape[1
]],
                  'rf__max_depth': [ 10, 20, 30, 40],
                  'rf__min_samples_split': [2, 5,  7],
                  "rf__min_samples_leaf": [1, 3, 10],
                  "rf__criterion": ["gini", "entropy"]}

gs_pipe_RF = GridSearchCV(estimator=pipe_RF,
                          param_grid=params_pipe_RF,
                          cv=cv_method,
                          refit=True,
                          n_jobs=-2,
                          scoring='accuracy',
                          verbose=1)

gs_pipe_RF.fit(x_train, y_train);
```

```
Fitting 5 folds for each of 360 candidates, totalling 1800 fits

[Parallel(n_jobs=-2)]: Using backend LokyBackend with 7 concurrent workers.
[Parallel(n_jobs=-2)]: Done  36 tasks      | elapsed:   31.8s
[Parallel(n_jobs=-2)]: Done 186 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-2)]: Done 436 tasks      | elapsed:  6.5min
[Parallel(n_jobs=-2)]: Done 786 tasks      | elapsed: 13.0min
[Parallel(n_jobs=-2)]: Done 1236 tasks      | elapsed: 22.2min
[Parallel(n_jobs=-2)]: Done 1786 tasks      | elapsed: 31.5min
[Parallel(n_jobs=-2)]: Done 1800 out of 1800 | elapsed: 31.7min finished
C:\Users\Vishwa\Anaconda3\lib\site-packages\ipykernel_launcher.py:16: DataCon
versionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples,), for example using ravel().
  app.launch_new_instance()
C:\Users\Vishwa\Anaconda3\lib\site-packages\sklearn\pipeline.py:267: DataConv
ersionWarning: A column-vector y was passed when a 1d array was expected. Ple
ase change the shape of y to (n_samples,), for example using ravel().
  self._final_estimator.fit(Xt, y, **fit_params)
```

Analysis - Grid Search Decision Tree

It can be seen from the 3d Surface plots below that the performance of models improves with increaseing values for number of features and maximum depth. 'Entropy' based methods performs slightly better than 'Gini index' based methods. we get highest results by using top 40 features as indicated by feature selection function.

In [20]:
```
gs_pipe_RF_sr = get_search_results(gs_pipe_RF)
gs_pipe_RF_sr.head()

fig = plt.figure(figsize = (8,6))
ax = fig.gca(projection='3d')
ax.plot_trisurf(gs_pipe_RF_sr['rfi_fs__n_features_'], gs_pipe_RF_sr['rf__max_depth'], gs_pipe_RF_sr['mean_score'],  linewidth=0.2)
_ = ax.set_xlabel("Number of Features")
_ = ax.set_ylabel("Maximum Depth")
_ = ax.set_zlabel("Score")
_ = ax.set_title("Surface plot of 'Number of Features' vs 'Maximum Depth' vs 'Mean Accuracy' ")
_ = ax.view_init(30, 210)
plt.show();
```



Results

Optimal values for parameters are

- rf__criterion = entropy,
- rf__max_depth = 20,
- rf__min_samples_leaf = 1
- rf__min_samples_split = 2
- rfi_fs__n*features* = 40

Highest Accuracy = 83.715 %

In [21]:
```
gs_pipe_RF.best_params_
```

Out[21]:
```
{'rf__criterion': 'entropy',
 'rf__max_depth': 40,
 'rf__min_samples_leaf': 1,
 'rf__min_samples_split': 2,
 'rfi_fs__n_features_': 40}
```

In [22]:
```
gs_pipe_RF.best_score_
```

Out[22]:
```
0.8410583111226679
```

Gaussian Naive Bayes Classifier

To tune the parameters of NB Classifier we will only use var_smoothing which is a variant of Laplace smoothing along with different number of features. we have used 40 iterations to search among 200 different values between 10 to (10 ^ -3).

In [23]:
```
# Naive Bayes
from sklearn.preprocessing import PowerTransformer
descriptives_powertransformed = PowerTransformer().fit_transform(descriptives)
x_train_NB, x_test_NB, y_train_NB, y_test_NB = train_test_split(descriptives_p
owertransformed, target, test_size = 0.20, random_state = 0)


# y_train_powertransformed = PowerTransformer().fit_transform(method = "box-co
x",y_train_NB)
```

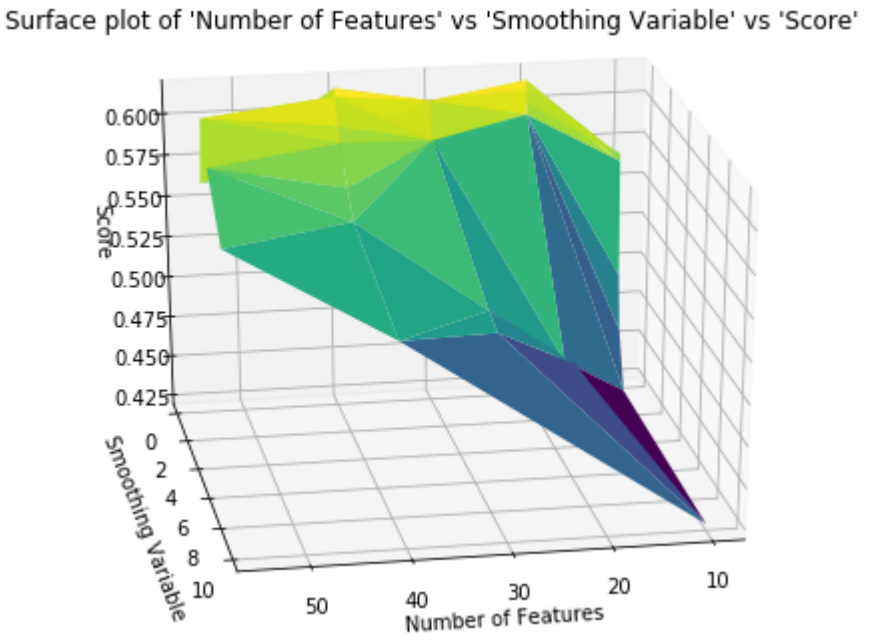
In [24]:
```
pipe_NB = Pipeline([('rfi_fs', FeatureSelectorRFI()),
                    ('nb', GaussianNB())])

params_pipe_NB = {'rfi_fs__n_features_': [10, 20,30, 40, descriptives.shape[1
]],
                  'nb__var_smoothing': np.logspace(1,-3, num=200)}

n_iter_search = 40
gs_pipe_NB = RandomizedSearchCV(estimator=pipe_NB,
                          param_distributions=params_pipe_NB,
                          cv=cv_method,
                          refit=True,
                          n_jobs=-2,
                          scoring='accuracy',
                          n_iter=n_iter_search,
                          verbose=1)

gs_pipe_NB.fit(x_train, y_train);
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits

[Parallel(n_jobs=-2)]: Using backend LokyBackend with 7 concurrent workers.
[Parallel(n_jobs=-2)]: Done  36 tasks      | elapsed:   33.4s
[Parallel(n_jobs=-2)]: Done 186 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-2)]: Done 200 out of 200 | elapsed:  2.8min finished
C:\Users\Vishwa\Anaconda3\lib\site-packages\ipykernel_launcher.py:16: DataCon
versionWarning: A column-vector y was passed when a 1d array was expected. Pl
ease change the shape of y to (n_samples,), for example using ravel().
  app.launch_new_instance()
C:\Users\Vishwa\Anaconda3\lib\site-packages\sklearn\utils\validation.py:761:
DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
```

Analysis - Grid Search Gaussian Naive Bayes

From the graph it can be observed that for 20 features score is highest, while it decreases slightly for more than 20 features. smaller value for smoothing parameter has positive impact on the model performance.

In [40]:
```
gs_pipe_NB_sr = get_search_results(gs_pipe_NB)
gs_pipe_NB_sr.head()

fig = plt.figure(figsize = (8,6))

ax = fig.gca(projection='3d')
ax.plot_trisurf(gs_pipe_NB_sr['rfi_fs__n_features_'], gs_pipe_NB_sr['nb__var_s
moothing'], gs_pipe_NB_sr['mean_score'],  cmap=plt.cm.viridis, linewidth=0.8)
_ = ax.set_xlabel("Number of Features")
_ = ax.set_ylabel("Smoothing Variable")
_ = ax.set_zlabel("Score")
_ = ax.set_title("Surface plot of 'Number of Features' vs 'Smoothing Variable'
vs 'Score' ")
_ = ax.view_init(25, 80)
plt.show();
```



Results

Optimal values for parameters are

- rfi_fs__n*features* = 20,
- nb__var_smoothing = 0.02121,

Highest Accuracy = 61.600 %

In [26]:
```
gs_pipe_NB.best_params_
```

Out[26]:
```
{'rfi_fs__n_features_': 40, 'nb__var_smoothing': 0.09329304026284686}
```

In [27]:
```
gs_pipe_NB.best_score_
```

Out[27]:
```
0.6160046804914516
```

Performance Comparision

We have obtained optimal parameters for all 4 classification models using hyper parameter tuning on training dataset. We will apply following steps to find out best performing model for our dataset.

- Fit all the otimized model pipeline on test data to retrieve predected target values
- Calculate 'accuracy score' for each model
- Perform pairwise ttests for all 4 models. We will use ttest_rel() method from stats package.

Making sure that model score is not influenced by 'lucky-split' or biased data splits, we will use 10 fold cross validation StratifiedKFold without repetation. Parallel processing can be used to speed up the procees. n_jobs= -2 indicates that all the cores except one will be used in the process allowing faster execution.

In [29]:
```
# KNN
cv_method_ttest = StratifiedKFold(n_splits=10, random_state=111)

cv_results_KNN = cross_val_score(estimator=gs_pipe_KNN.best_estimator_,
                                 X=x_test,
                                 y=y_test,
                                 cv=cv_method_ttest,
                                 n_jobs=-2,
                                 scoring='accuracy')
KNN_val = cv_results_KNN.mean()
```

In [33]:
```
# Decision Tree
cv_results_DT = cross_val_score(estimator=gs_pipe_DT.best_estimator_,
                                X=x_test,
                                y=y_test,
                                cv=cv_method_ttest,
                                n_jobs=-2,
                                scoring='accuracy')
DT_val = cv_results_DT.mean()
```

In [34]:
```
# Random Forest
cv_results_RF = cross_val_score(estimator=gs_pipe_RF.best_estimator_,
                                X=x_test,
                                y=y_test,
                                cv=cv_method_ttest,
                                n_jobs=-2,
                                scoring='accuracy')
RF_val = cv_results_RF.mean()
```

In [35]:
```
# Naive Bayes
cv_results_NB = cross_val_score(estimator=gs_pipe_NB.best_estimator_,
                                X=x_test_NB,
                                y=y_test_NB,
                                cv=cv_method_ttest,
                                n_jobs=-2,
                                scoring='accuracy')
NB_val = cv_results_NB.mean()
```

In [36]:
```
# Score table
accuracy_vals = [KNN_val, DT_val, RF_val, NB_val]
tables_cols = ["KNN", "DT", "RF", "NB"]
table = pd.DataFrame(accuracy_vals, tables_cols)
table.columns = ["Accuracy Score"]
table.head()
```

Out[36]:

	Accuracy Score
KNN	0.733979
DT	0.704130
RF	0.760017
NB	0.569465

For pairwise performance assesment of classifiers, we will perform pairwise statistical t-test determining if difference between performance of any two model on our dataset is statistically significant or not. In following code segment, t-test will be used to find statistical significance between following pair or classification models:

- KNN vs DT
- KNN vs RF
- KNN vs NB
- DT vs RF
- DT vs NB
- RF vs NB

In [37]:
```
cv_results = [cv_results_KNN,cv_results_DT, cv_results_RF, cv_results_NB]
cv_models = ['KNN', 'DT', ' RF', 'NB']
for i in range(0,3):
    for j in range(i+1, 4):
        print(cv_models[i] +' vs ' + cv_models[j])
        print(stats.ttest_rel(cv_results[i], cv_results[j]))
```

```
KNN vs DT
Ttest_relResult(statistic=3.939951950887926, pvalue=0.003406210321527855)
KNN vs  RF
Ttest_relResult(statistic=-3.272775876524596, pvalue=0.00963986781055809)
KNN vs NB
Ttest_relResult(statistic=20.085430111430608, pvalue=8.744494000563827e-09)
DT vs  RF
Ttest_relResult(statistic=-6.625118228742057, pvalue=9.645051597749004e-05)
DT vs NB
Ttest_relResult(statistic=24.52039420188273, pvalue=1.495283831813947e-09)
 RF vs NB
Ttest_relResult(statistic=20.62874538871195, pvalue=6.909182893390744e-09)
```

Here, for statistical t-test , pvalue lower than 0.05 implies that models have statistically significant difference in terms of performace on test data with best estimators. For such cases, it can be derived that one model outperforms other. Random Forest is statistically best performing classification model(based on Accuracy Score) when applied on test data.

Let's create classification report of test data prediction using classification_report from sklearn.metrics.

In [38]:
```
y_pred_KNN = gs_pipe_KNN.predict(x_test)
#KNN_valas = metrics.f1_score(y_test, y_pred_KNN, average='micro')

y_pred_DT = gs_pipe_DT.predict(x_test)
#DT_valas = metrics.accuracy_score(y_test, y_pred_DT)

y_pred_RF = gs_pipe_RF.predict(x_test)
#RF_valas = metrics.accuracy_score(y_test, y_pred_RF)

y_pred_NB = gs_pipe_NB.predict(x_test_NB)
#NB_valas = metrics.accuracy_score(y_test_NB, y_pred_NB)
```

In [39]:
```
print(metrics.classification_report(y_test, y_pred_KNN))
print(metrics.classification_report(y_test, y_pred_DT))
print(metrics.classification_report(y_test, y_pred_RF))
print(metrics.classification_report(y_test_NB, y_pred_NB))
```

	precision	recall	f1-score	support
1	0.75	0.73	0.74	542
2	0.72	0.67	0.69	524
3	0.80	0.74	0.77	570
4	0.89	0.94	0.92	535
5	0.89	0.94	0.92	562
6	0.77	0.80	0.79	561
7	0.94	0.95	0.95	552
micro avg	0.83	0.83	0.83	3846
macro avg	0.82	0.83	0.82	3846
weighted avg	0.82	0.83	0.82	3846

	precision	recall	f1-score	support
1	0.66	0.65	0.65	542
2	0.61	0.60	0.61	524
3	0.76	0.74	0.75	570
4	0.91	0.95	0.93	535
5	0.87	0.91	0.89	562
6	0.77	0.76	0.77	561
7	0.93	0.91	0.92	552
micro avg	0.79	0.79	0.79	3846
macro avg	0.79	0.79	0.79	3846
weighted avg	0.79	0.79	0.79	3846

	precision	recall	f1-score	support
1	0.74	0.74	0.74	542
2	0.74	0.65	0.69	524
3	0.81	0.82	0.81	570
4	0.91	0.97	0.94	535
5	0.89	0.96	0.92	562
6	0.84	0.81	0.82	561
7	0.93	0.95	0.94	552
micro avg	0.84	0.84	0.84	3846
macro avg	0.84	0.84	0.84	3846
weighted avg	0.84	0.84	0.84	3846

	precision	recall	f1-score	support
1	0.52	0.44	0.48	542
2	0.41	0.56	0.47	524
3	0.41	0.85	0.56	570
4	0.82	0.60	0.69	535
5	0.56	0.50	0.53	562
6	0.54	0.07	0.13	561
7	0.88	0.82	0.85	552
micro avg	0.55	0.55	0.55	3846
macro avg	0.59	0.55	0.53	3846
weighted avg	0.59	0.55	0.53	3846

Accuracy score gives us information about how often our model prediction is correct telling whether model is being trained correctly or not. It works well with only balanced data. This constraint is not our concern as we have already created balanced data for training purpose during phase 1 of project.

Classification report can be used to select best classifier according to nature of problem. If false negetive in our data has devastating effect, then choosing a model with lowest recall makes sense. In our case, in order to correctly find a specific cover type in dense forest we should use precision score. As it might cause very high expense in terms of resouces and time to reach to specific area. Hence, it is of most improtant for a model to predict correct data. Random forest has highest precision value among all 4 classification models. Therefore, it should be chosen over other models.

Limitations and Proposed Solutions

In this analysis, we have not used entire dataset available to perform analysis. Main reason behind this is highly imbalanced nature of the target data. Cosidering span of the project, we chose to proceed with undersampling of data. As an advanced step, data can be oversampled and fitted to models to evaluate performance enhancement aligned with number of increased instances.

We have performed hyperparameter tuning with k-fold cross validation. However, due to time and hardware limitations, we could not perform exhaustive grid search cross validation with broader range of hyperparameter space. To overcome this issue, this study can be extended on system with better hardware capabilities.

Moreoever, simple K-fold cross validation(without any repetation) is used here to control processing time. To achieve better optimized results, repeated K-fold can be implemented in pipeline.

In future, this study can be further extended by implementing advanced neural netowrks and deep learning techniques.

Summary

In this report, we mainly focused on modeling and evaluating a classification problem in pipeline. Data used is already preprocessed in previous phase of the project onn which we applied minmax scaling. We calculated feature importance score and listed 10 most important features contributing to target feature using RandomForestImportance(RFI). Data is divided into 80:20 train:test splits. We stacked feature selection and hyperparameter tuning in a pipeline to obtain optimal parameters. In Pipeline, we have added different number of set of features for selection, diffrent value for relevent parameters of model. Main objective of this entire project is to improve and evaluate performance of four classification models namely K-Nearest Neighbours(KNN), Decision Tree(DT), Random Forest search(RF) and Naive Bayes(NB)- Gaussian. GridSearchCV and RandomSearchCV metods are used on training dataset to otimize hyperparameter for all models. As a final evaluation step, we performed pairwise t-test and generated cross value score by fitting each model on test dataset with achieved bets hyperparameters.

From the study we found that, for our dataset best performing classification model is Random Forest with entropy criterion, maximum depth of 20, minimum sample leaf equals 1, minimum sample spit equals 2 and all the feature selected. KNN and RF works well with all the features selected. Where as, DT and NB are efficient with partial feature set of 20 and 40 respectively. Classification report depicts that Random Forest outperforms other models as it has highest value for scores generated. Hence, it is the most efficient classifier for our dataset.

References

- Jock A. Blackard and Colorado State University : UCI Machine Learning Repository : covertype dataset https://archive.ics.uci.edu/ml/datasets/covertype (https://archive.ics.uci.edu/ml/datasets/covertype)
- Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011 : https://scikit-learn.org/stable/ (https://scikit-learn.org/stable/)
- Ploting graphs : https://python-graph-gallery.com/ (https://python-graph-gallery.com/)