Assignment 3 Part 2

CS4172 Machine Learning Lab

Name: Abhiroop Mukherjee

Enrolment Number: 510519109

Task 4

Download the Forest Cover Type dataset (https://www.kaggle.com/uciml/forest-cover-type-dataset) and pre-process the dummy variables to create training, test, and development set. Reduce the train data size if the system unable to process the whole dataset.

In [ ]:
```
import pandas as pd

_FILE_PATH = './../ML_DRIVE/Assign_3/covtype/covtype.csv'

cov_df = pd.read_csv(_FILE_PATH)

cov_df
```

Out[ ]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshade_Noo
0	2596	51	3	258	0	510	221	23
1	2590	56	2	212	-6	390	220	23
2	2804	139	9	268	65	3180	234	23
3	2785	155	18	242	118	3090	238	23
4	2595	45	2	153	-1	391	220	23
...	...	...	...	...	...	...	...	.
581007	2396	153	20	85	17	108	240	23
581008	2391	152	19	67	12	95	240	23
581009	2386	159	17	60	7	90	236	24
581010	2384	170	15	60	5	90	230	24
581011	2383	165	13	60	4	67	231	24

581012 rows × 55 columns

In [ ]:
```
cov_df.columns
```

Out[ ]:
```
Index(['Elevation', 'Aspect', 'Slope', 'Horizontal_Distance_To_Hydrology',
       'Vertical_Distance_To_Hydrology', 'Horizontal_Distance_To_Roadways',
       'Hillshade_9am', 'Hillshade_Noon', 'Hillshade_3pm',
       'Horizontal_Distance_To_Fire_Points', 'Wilderness_Area1',
       'Wilderness_Area2', 'Wilderness_Area3', 'Wilderness_Area4',
       'Soil_Type1', 'Soil_Type2', 'Soil_Type3', 'Soil_Type4', 'Soil_Type5',
       'Soil_Type6', 'Soil_Type7', 'Soil_Type8', 'Soil_Type9', 'Soil_Type10',
       'Soil_Type11', 'Soil_Type12', 'Soil_Type13', 'Soil_Type14',
       'Soil_Type15', 'Soil_Type16', 'Soil_Type17', 'Soil_Type18',
       'Soil_Type19', 'Soil_Type20', 'Soil_Type21', 'Soil_Type22',
       'Soil_Type23', 'Soil_Type24', 'Soil_Type25', 'Soil_Type26',
       'Soil_Type27', 'Soil_Type28', 'Soil_Type29', 'Soil_Type30',
       'Soil_Type31', 'Soil_Type32', 'Soil_Type33', 'Soil_Type34',
       'Soil_Type35', 'Soil_Type36', 'Soil_Type37', 'Soil_Type38',
       'Soil_Type39', 'Soil_Type40', 'Cover_Type'],
      dtype='object')
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler

def standardize(df: "pd.DataFrame", col_name: "str") -> "pd.DataFrame":
    scaler = StandardScaler()
```

```
    df[[col_name]] = pd.DataFrame(
        data=scaler.fit_transform(df[[col_name]]),
        index=df.index,
        columns=[col_name]
    )
    return df
```

In [ ]:
```
_columns_to_scale = ['Elevation', 'Aspect', 'Slope', 'Horizontal_Distance_To_Hydrology',
                     'Vertical_Distance_To_Hydrology', 'Horizontal_Distance_To_Roadways',
                     'Hillshade_9am', 'Hillshade_Noon', 'Hillshade_3pm',
                     'Horizontal_Distance_To_Fire_Points']

for _col in _columns_to_scale:
    cov_df = standardize(cov_df, _col)

cov_df
```

Out[ ]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshad
0	-1.297805	-0.935157	-1.482820	-0.053767	-0.796273	-1.180146	0.330743	0
1	-1.319235	-0.890480	-1.616363	-0.270188	-0.899197	-1.257106	0.293388	0
2	-0.554907	-0.148836	-0.681563	-0.006719	0.318742	0.532212	0.816364	0
3	-0.622768	-0.005869	0.520322	-0.129044	1.227908	0.474492	0.965786	0
4	-1.301377	-0.988770	-1.616363	-0.547771	-0.813427	-1.256464	0.293388	0
...	...	...	...	...	...	...	...	
581007	-2.012130	-0.023740	0.787408	-0.867697	-0.504653	-1.437962	1.040496	0
581008	-2.029988	-0.032675	0.653865	-0.952383	-0.590424	-1.446299	1.040496	0
581009	-2.047847	0.029873	0.386780	-0.985317	-0.676194	-1.449506	0.891075	0
581010	-2.054990	0.128163	0.119694	-0.985317	-0.710502	-1.449506	0.666942	1
581011	-2.058562	0.083486	-0.147392	-0.985317	-0.727656	-1.464256	0.704298	1

581012 rows × 55 columns

In [ ]:
```
cov_df[['Cover_Type']].value_counts()
```

Out[ ]:
```
Cover_Type
2              283301
1              211840
3               35754
7               20510
6               17367
5                9493
4                2747
dtype: int64
```

In [ ]:
```
# solving imbalance by selection based under-sampling

# https://imbalanced-learn.org/stable/references/generated/imblearn.datasets.make_imbalance.html
from imblearn.datasets import make_imbalance

_X = cov_df.drop('Cover_Type', axis=1)
_y = cov_df[['Cover_Type']]
_min_sample_size = cov_df[['Cover_Type']].value_counts().min()

X_bal, y_bal = make_imbalance(
    _X,
    _y,
    sampling_strategy={
        x: _min_sample_size for x in _y.iloc[:,0].unique()
    }
)
```

In [ ]:
```
y_bal.value_counts()
```

Out[ ]:
```
Cover_Type
1              2747
2              2747
3              2747
4              2747
5              2747
6              2747
7              2747
dtype: int64
```

In [ ]:
```
# 80% as train
# 10% as validation
# 10% as train

from sklearn.model_selection import train_test_split

X_train, _X_rest, y_train, _y_rest = train_test_split(X_bal, y_bal, train_size=0.8)
X_val, X_test, y_val, y_test = train_test_split(_X_rest, _y_rest, train_size=0.5)
```

Task 5

Apply multi-class classification in SVM using Forest Cover Type dataset.

In [ ]:
```python
# https://scikit-learn.org/stable/modules/svm.html#svm
# chose LinearSVC cause no mention of kernel to be used
# and LinearSVC is the fastest


# https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, f1_score

_model = LinearSVC(max_iter=1000000).fit(X_train, y_train.iloc[:, 0])

_y_predict = _model.predict(X_val)
_accuracy = accuracy_score(y_val, _y_predict)
# macro takes mean for all the individual f1_scores for different labels
_f1 = f1_score(y_val, _y_predict, average='macro')

print(f"default accuracy = {_accuracy}")
print(f"default f1 = {_f1}")
```

```
default accuracy = 0.6687467498699948
default f1 = 0.659146923194361
```

In [ ]:
```python
# hyper parameter tuning

def svm_train(
    X_train: "pd.DataFrame",
    X_val: "pd.DataFrame",
    y_train: "pd.DataFrame",
    y_val: "pd.DataFrame",
    penalty='l2',
    loss="squared_hinge",
    dual=False,
    tol=1e-4,
    C=1.0
) -> "LinearSVC":
    '''
    Wrapper Function for sklearn.svm.LinearSVC
    See: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC
    '''

    model = LinearSVC(
        penalty=penalty,
        loss=loss,
```

```
        dual=dual,
        tol=tol,
        C=C,
        max_iter=1000000
    ).fit(X_train, y_train.iloc[:, 0])

    y_predict = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_predict)
    f1 = f1_score(y_val, y_predict, average='macro')

    return [penalty, loss, tol, C, accuracy, f1]
```

In [ ]:
```
# Test 1: L1 vs L2

_result = []

_result.append(svm_train(X_train, X_val, y_train, y_val, penalty='l1'))
_result.append(svm_train(X_train, X_val, y_train, y_val, penalty='l2'))

_df = pd.DataFrame(
    data=_result,
    columns=['Penalty', "Loss", "Tolerance",
             "C", "Accuracy", "f1"]
)
print(_df)

best_penalty = _df.sort_values(['f1', 'Accuracy'], ascending=False)\
    .iloc[0,:]['Penalty']
print(f"best penalty = {best_penalty}")

# Result Changes use case basis
# https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization#:~:text=The%20differences%20between%20L1%20and,of%20squares%20o
```

```
  Penalty           Loss  Tolerance    C  Accuracy        f1
0      l1  squared_hinge     0.0001  1.0  0.669267  0.659771
1      l2  squared_hinge     0.0001  1.0  0.668747  0.659147
best penalty = l1
```

In [ ]:
```
if best_penalty == 'l1':
    # as l1 does not support 'hinge'
    best_loss = 'squared_hinge'
    print(f"best loss = {best_loss}")
else:
    # Test 2: ‘hinge’ vs ‘squared_hinge’
    _result = []

    # dual= True used for compatibility
    _result.append(svm_train(X_train, X_val, y_train, y_val,
```

```
                            penalty=best_penalty, loss="hinge", dual=True))
    _result.append(svm_train(X_train, X_val, y_train, y_val,
                            penalty=best_penalty, loss="squared_hinge", dual=True))

    _df = pd.DataFrame(
        data=_result,
        columns=['Penalty', "Loss", "Tolerance",
                 "C", "Accuracy", "f1"]
    )
    print(_df)
    best_loss = _df.sort_values(['f1', 'Accuracy'], ascending=False)\
        .iloc[0,:]['Loss']
    print(f"best loss = {best_loss}")
```

```
best loss = squared_hinge
```

In [ ]:
```
# Test 3: tolerance test
_result = []
_tolerances = [1e-5, 3.3333e-5, 6.6666e-5, 1e-4, 3.333e-4, 6.666e-4, 1e-3]

for tol in _tolerances:
    _result.append(svm_train(X_train, X_val, y_train, y_val,
                  penalty=best_penalty, loss=best_loss,
                  dual=False, tol=tol))

_df = pd.DataFrame(
    data=_result,
    columns=['Penalty', "Loss", "Tolerance",
             "C", "Accuracy", "f1"]
)
print(_df)
best_tol = _df.sort_values(['f1', 'Accuracy'], ascending=False)\
    .iloc[0,:]['Tolerance']
print(f"best tolerance = {best_tol}")
```

```
  Penalty           Loss  Tolerance    C  Accuracy        f1
0      l1  squared_hinge   0.000010  1.0  0.669267  0.659771
1      l1  squared_hinge   0.000033  1.0  0.669267  0.659771
2      l1  squared_hinge   0.000067  1.0  0.669267  0.659771
3      l1  squared_hinge   0.000100  1.0  0.669267  0.659771
4      l1  squared_hinge   0.000333  1.0  0.669267  0.659771
5      l1  squared_hinge   0.000667  1.0  0.669267  0.659846
6      l1  squared_hinge   0.001000  1.0  0.669787  0.660469
best tolerance = 0.001
```

In [ ]:
```
# Test 5: C values

_result = []
_c_values = [0.3333, 0.6666, 1, 3.3333, 6.6666, 10]
```

```
for c in _c_values:
    _result.append(svm_train(X_train, X_val, y_train, y_val,
                             penalty=best_penalty, loss=best_loss,
                             dual=False, tol=best_tol,C=c))

_df = pd.DataFrame(
    data=_result,
    columns=['Penalty', "Loss", "Tolerance",
             "C", "Accuracy", "f1"]
)

print(_df)
best_C = _df.sort_values(['f1', 'Accuracy'], ascending=False)\
    .iloc[0,:]['C']
print(f"best tolerance = {best_C}")
```

```
  Penalty           Loss  Tolerance        C  Accuracy        f1
0      l1  squared_hinge      0.001   0.3333  0.670307  0.660508
1      l1  squared_hinge      0.001   0.6666  0.670307  0.660975
2      l1  squared_hinge      0.001   1.0000  0.670307  0.660891
3      l1  squared_hinge      0.001   3.3333  0.669787  0.660383
4      l1  squared_hinge      0.001   6.6666  0.669787  0.660265
5      l1  squared_hinge      0.001  10.0000  0.669787  0.660377
best tolerance = 0.6666
```

In [ ]:
```
# End Result after using validation dataset for all hyper-parameters

model = LinearSVC(
        penalty=best_penalty,
        loss=best_loss,
        dual=False,
        tol=best_tol,
        C=best_C,
        max_iter=1000000
    ).fit(X_train, y_train.iloc[:, 0])

y_predict = model.predict(X_test)
accuracy = accuracy_score(y_test, y_predict)
f1 = f1_score(y_test, y_predict, average='macro')

print(f"Test Accuracy: {accuracy}")
print(f"Test F1: {f1}")
```

```
Test Accuracy: 0.6713468538741549
Test F1: 0.6688651003031573
```

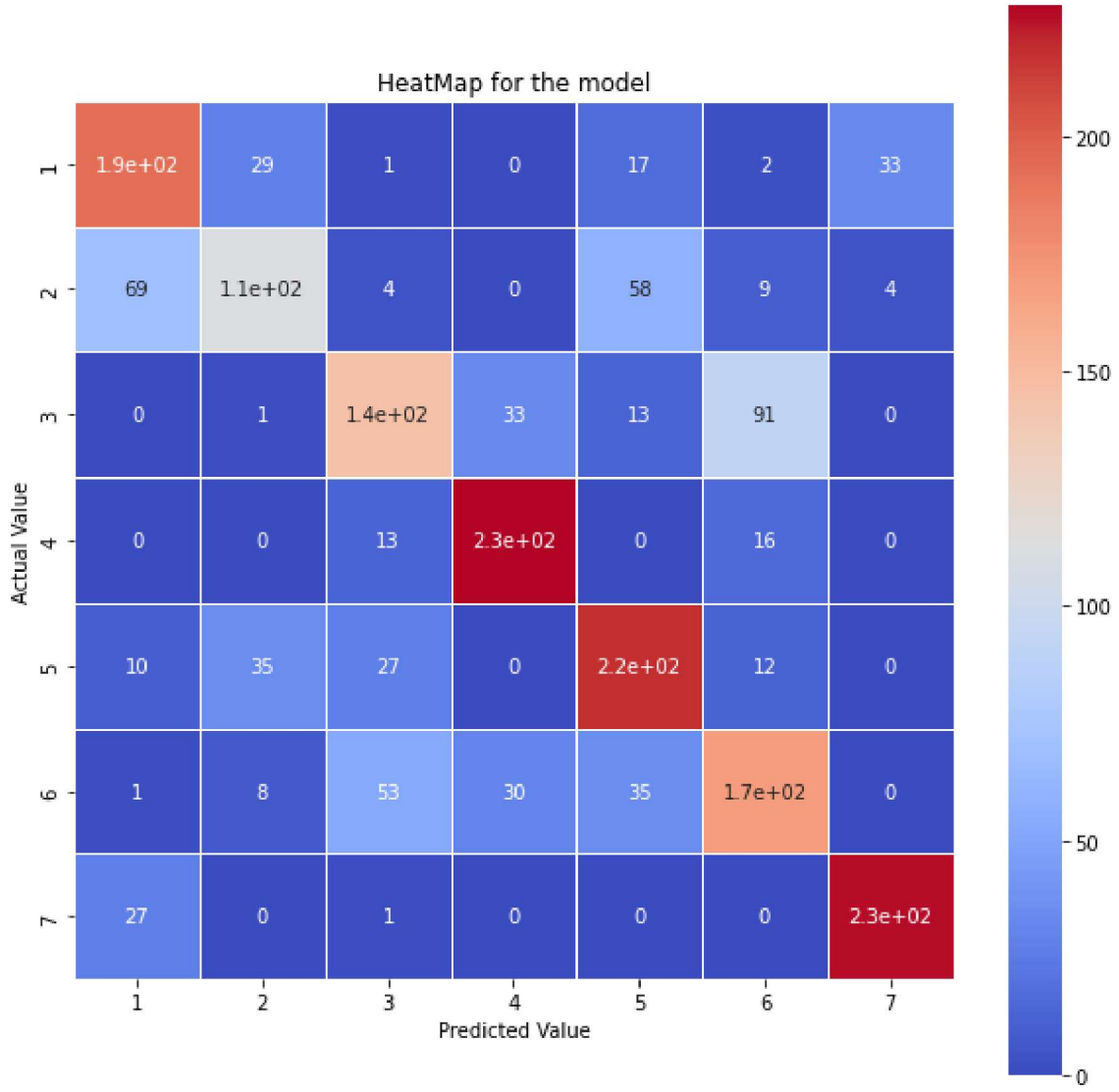
Task 6

Plot and Analyze the Confusion matrix for the above applied SVM method.

In [ ]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

matrix = confusion_matrix(y_test, y_predict)
fig = plt.figure(figsize=(10,10))
sns.heatmap(
    matrix,
    xticklabels=range(1,8),
    yticklabels=range(1,8),
    linewidth=0.5,
    cmap='coolwarm',
    annot=True,
    cbar=True,
    square=True)
plt.title('HeatMap for the model')
plt.ylabel('Actual Value')
plt.xlabel('Predicted Value')
plt.show()
```



Task 7

Consider only two features and three classes and train Logistic Regression 3-class Classifier (Any three-class) to show the training and test area in a 2-D plane, using matplotlib.

In [ ]:
```
X_train
```

Out[ ]:

	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshade_
14788	-3.222912	-0.917286	2.389922	0.449646	-0.899197	-1.257106	0.218677	-3.5
508	-0.272748	-1.158544	-0.548020	0.063852	-0.092955	0.073659	0.106611	-0.2
14820	-2.480013	-0.586674	-1.482820	-0.561885	-0.007185	-0.538172	0.442810	0.5
701	0.595157	1.245096	-0.948649	-0.105520	-0.607578	-0.698505	-0.416366	0.7
11605	-0.072737	1.665063	-0.948649	-1.126460	-0.813427	1.552572	-0.229588	0.3
...	...	...	...	...	...	...	...	
1334	0.648731	1.531031	0.653865	0.651952	-0.161572	-1.272498	-1.574384	-0.4
4099	-0.529905	1.396999	0.520322	-0.985317	-0.693348	-0.162351	-1.574384	0.0
15198	-1.762117	-1.390866	0.520322	-0.547771	-0.075801	-1.314826	-0.827275	-1.0
5459	-0.540620	-0.801125	-1.082191	-0.952383	-0.676194	0.630977	0.554876	0.1
12719	-1.079936	-0.908351	0.119694	-1.126460	-0.830581	-0.605512	0.592231	-0.8

15383 rows × 54 columns

In [ ]:

```
#  taking first two features

subset_X_train = X_train.iloc[:, 0:2]
subset_y_train = y_train

subset_train = subset_X_train.join(subset_y_train)
subset_train = subset_train[subset_train['Cover_Type'].isin([1,2,3])]

subset_train
```

Out[ ]:

	Elevation	Aspect	Cover_Type
508	-0.272748	-1.158544	1
701	0.595157	1.245096	1
2217	0.723735	1.423805	1
4246	-0.226317	-1.113866	2
632	0.673733	-0.256062	1
...	...	...	...
5314	-0.190601	-0.693899	2
6794	-2.937182	-0.559868	3
1334	0.648731	1.531031	1
4099	-0.529905	1.396999	2
5459	-0.540620	-0.801125	2

6609 rows × 3 columns

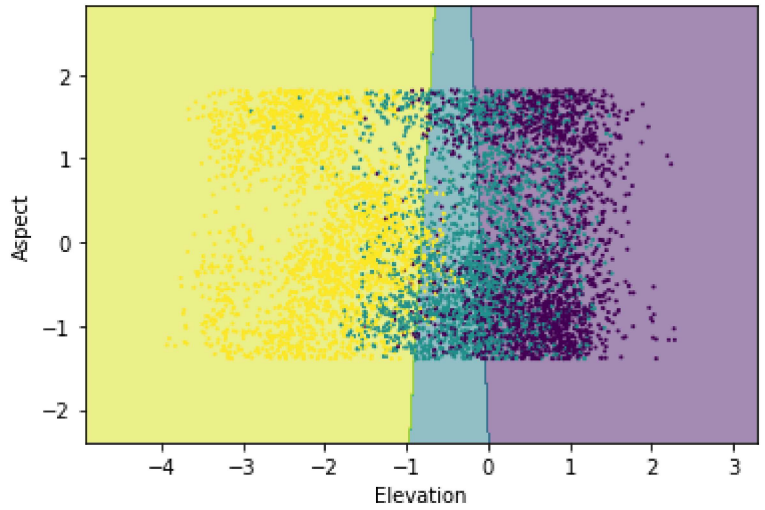
In [ ]:
```
model = LinearSVC().fit(
    subset_train.iloc[:, 0:2],
    subset_train.iloc[:, 2]
)
```

In [ ]:
```
from sklearn.inspection import DecisionBoundaryDisplay

disp = DecisionBoundaryDisplay.from_estimator(
    model,
    subset_train.iloc[:, 0:2],
    xlabel='Elevation',
    ylabel='Aspect',
    alpha=0.5,
    grid_resolution=5000
)

disp.ax_.scatter(
    subset_train['Elevation'],
    subset_train['Aspect'],
    c=subset_train['Cover_Type'],
    s=1
)
```

Out[ ]: <matplotlib.collections.PathCollection at 0x1b40e51d420>



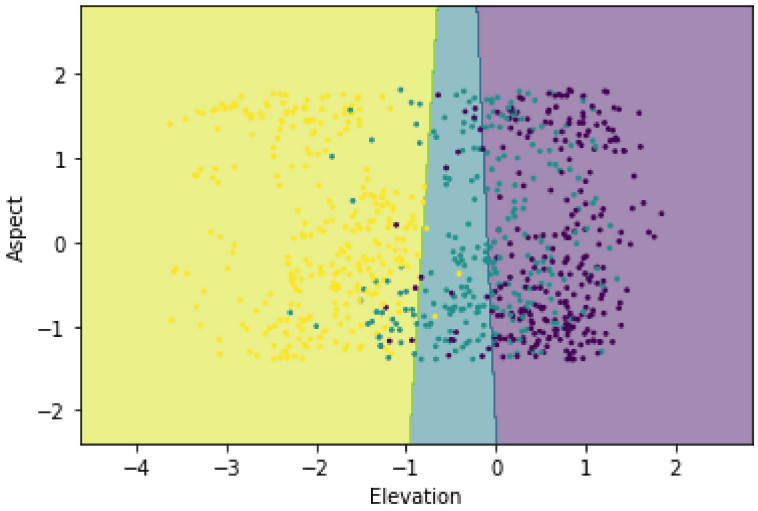
In [ ]:
```
subset_X_test = X_test.iloc[:, 0:2]
subset_y_test = y_test

subset_test = subset_X_test.join(subset_y_test)
subset_test = subset_test[subset_test['Cover_Type'].isin([1,2,3])]

disp = DecisionBoundaryDisplay.from_estimator(
    model,
    subset_test.iloc[:, 0:2],
    xlabel='Elevation',
    ylabel='Aspect',
    alpha=0.5,
    grid_resolution=5000
)

disp.ax_.scatter(
    subset_test['Elevation'],
    subset_test['Aspect'],
    c=subset_test['Cover_Type'],
    s=3
)
```

Out[ ]: <matplotlib.collections.PathCollection at 0x1b40e5e87c0>


Aspect
Elevation
2
1
0
−1
−2
−4
−3
−2
−1
0
1
2