Final Programming Report

The target of the final programming project is about satellite image classification. To achieve this target, I build a support vector classifier and a random forest classifier to detect various land cover classes. Both of the classifier are trained by 40,000 training datasets. The prediction accuracy of the support vector classifier tested by 100,000 test dataset is up to 98.8%.

The workflow of the project is as follows: (1). Raw dataset Loading and analysis (2). Preprocessing the original dataset into a suitable format for machine learning algorithm, including feature extraction and normalization. (3) Tune hyperparameters using cross validation (4) Train classifiers using the optimal hyperparameters and model testng (5) Data visualization

1.Raw dataset loading and analysis

First of all, we load the raw data using the following code：

```
from scipy.io import loadmat
data = loadmat('/classes/ece2720/fpp/sat-4-full.mat')
trainx = data['train_x']
trainy = data['train_y']
```

The features of the training data is a 4-layer array. The first layer is the x axis of the pixel, including 28 items. Within each item in the first layer, there are also 28 items in the second layer, which represents y axis of the pixel. The first layer and the second layer determine a specific pixel. The third layer contains 4 channels, which are red, green, blue and NIR components. Within the 4 items, it is the fourth layer which indicates the 400,000 images.

The labels of training data is a 4-layer array. The first layer indicates the type of land and the second layer indicates the 400,000 images.

For the test dataset, it has the same format as training data.

2.Data preprocessing

During the data preprocessing, my target is to expand the 4-dimension training data into 2 dimension dataframe. By referring to the literature, we know that it is valuable to convert the RGB features into HSV for the satellite image classification. During the feature extracting, I calculate the mean value and standard deviation of H, S, V, NIR separately in each image. Therefore, each image has 8 features, which can be used in later modeling training. The code is as follows

```
import numpy as np
import colorsys
```

```
def RGB_HSV_converter(data,x):
    """This function convert all the RBG values into HSV format in an image,
    x is the index of the image"""
    red = annots[data][:,:,0,x]
    green = annots[data][:,:,1,x]
    blue = annots[data][:,:,2,x]
    NIR = annots[data][:,:,3,x]
    red = red.flatten()
    green = green.flatten()
    blue = blue.flatten()
    NIR = NIR.flatten()
    hue = np.ones(784)
    saturation = np.ones(784)
    value = np.ones(784)
    for i in range(784):
        HSV = colorsys.rgb_to_hsv(red[i]/float(255),green[i]/float(255),blue[i]/float(255))
        hue[i] = HSV[0]
        saturation[i] = HSV[1]
        value[i] = HSV[2]

    hue_mean = np.mean(hue)
    hue_std = np.std(hue)
    saturation_mean = np.mean(saturation)
    saturation_std = np.std(saturation)
    value_mean = np.mean(value)
    value_std = np.std(value)
    NIR_mean = np.mean(NIR)
    NIR_std = np.std(NIR)
    return [hue_mean, hue_std, saturation_mean, saturation_std, value_mean, value_std, NIR_mean, NIR_std]

import pandas as pd
new_feature = []
for i in range(400):
    new_feature.append(RGB_HSV_converter('train_x',i))
# create the DataFrame
df = pd.DataFrame(new_feature, columns = ['hue_mean', 'hue_std', 'saturation_mean', 'saturation_std', 'value_mean',
'value_std', 'NIR_mean', 'NIR_std'])
```

Table.1 Preprocessed data without normalization

	hue_mean	hue_std	saturation_mean	saturation_std	value_mean	value_std	NIR_mean	NIR_std
0	0.302352	0.221282	0.099610	0.065520	0.443607	0.083579	147.093112	26.336379
1	0.094648	0.111863	0.233111	0.046380	0.651581	0.067891	145.095663	12.551213
2	0.363837	0.231357	0.184358	0.144247	0.351676	0.108887	121.929847	41.970032
3	0.183581	0.226149	0.074254	0.066595	0.433193	0.118928	127.181122	37.392367
4	0.133382	0.113695	0.134510	0.056476	0.422039	0.054106	135.001276	8.643964
5	0.094021	0.010007	0.300761	0.036946	0.601586	0.031434	188.901786	6.141968
6	0.505201	0.298407	0.256446	0.131016	0.518362	0.094913	137.707908	33.141589
7	0.339751	0.205087	0.185481	0.128080	0.371028	0.106693	144.323980	37.117534
8	0.120816	0.009858	0.228213	0.017643	0.767197	0.049470	186.831633	4.783285

After exporting the CSV document of the training dataset, I find the data of NIR features in very large, which probably dominate their other feature during model training. Therefore, NIR_mean and NIR_std are normalized to lie in the range [0, 1]. This is done using the following codes:

```
import pandas as pd
from sklearn import preprocessing

x = df.iloc[:,6:8].values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)

df.iloc[:,6:8] = x_scaled
```

Table.2 Preprocessed data after normalization

	hue_mean	hue_std	saturation_mean	saturation_std	value_mean	value_std	NIR_mean	NIR_std
0	0.302352	0.221282	0.099610	0.065520	0.443607	0.083579	0.604857	0.402514
1	0.094648	0.111863	0.233111	0.046380	0.651581	0.067891	0.595682	0.180000
2	0.363837	0.231357	0.184358	0.144247	0.351676	0.108887	0.489263	0.654866
3	0.183581	0.226149	0.074254	0.066595	0.433193	0.118928	0.513386	0.580975
4	0.133382	0.113695	0.134510	0.056476	0.422039	0.054106	0.549310	0.116930
5	0.094021	0.010007	0.300761	0.036946	0.601586	0.031434	0.796918	0.076544
6	0.505201	0.298407	0.256446	0.131016	0.518362	0.094913	0.561744	0.512361
7	0.339751	0.205087	0.185481	0.128080	0.371028	0.106693	0.592137	0.576539
8	0.120816	0.009858	0.228213	0.017643	0.767197	0.049470	0.787408	0.054613

Now we can see the format of the data is extremely useful for model training.

3 Hyperparameter tuning

At the first attempt, I choose the SVM algorithms because it has been well elaborated in the class and I am more comfortable using to design a satisfactory classifier for the satellite image dataset. I employ the default parameters of the SVC function in the sklearn package to train the classifier and model score to evaluate its performance. However, the result is not good. The model's prediction accuracy scores of the test data is only 87%. I have to tune the model's hyperparameter to design a better classifier.

After reading the sklearn documentation, I decide to use GridSearchCV function to select the hyperparameter. The GridSearchCV is used for exhaustive search over specified parameter values for an estimator s.t. SVC. The parameters of the estimator are optimized by cross-validated grid-search over a parameter grid. Before we employ the GridSearchCV function, I need to talk about the important hyperparameter of the SVM.

In this project, I only consider some most important hyperparameters, which are C, kernel, gamma.

3.1 C

C is the penalty factor and it determined the generalization ability of the support vector classifier. A larger C represents the greater the penalty for misclassification sample. The accuracy of the training sample becomes higher with the increasing of the C value, while the model's generalization ability is decreasing, that is, the classification accuracy of the test data decreases. On the contrary, A smaller C will allow more misclassified samples in the training samples. Therefore, the model will have stronger generalization ability.

3.2 kernel

The kernel function is used to map the original non-linear observations into a higher-dimensional space in which they become separable. In this project. we consider the 'linear' and 'rbf' kernels.

3.3 gamma

Gamma is the kernel coefficients and it is only valid for 'RBF', 'poly', and 'sigmod' kernels.

I tried different sets of kernel, C and gamma to find the optimal combination. At the beginning I set a broad range of the hyperparameters. When I choose linear kernel, I set the range of C between 0.1-1000. When kernel is rbf, the range of C is 0.1-10000 and range of gamma is 0.01-1000. The code is as follows:

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
param_grid = [
  {'C': [0.1, 1, 10, 100, 1000, 10000], 'kernel': ['linear']},
  {'C': [0.1, 1, 10, 100, 1000, 10000], 'gamma': [0.01, 0.1, 1, 10, 100, 1000], 'kernel': ['rbf']},
 ]
svc = svm.SVC()
grid = GridSearchCV(estimator= svc,
                    param_grid = param_grid,
                    scoring = 'accuracy', )
```

```
grid_results = grid.fit(X, y)
print('Best Score: ', grid_results.best_score_)
print('Best Params: ', grid_results.best_params_)
```

Since the classifiers with the rbf kernel performances better than those with linear kernel under the satellite image dataset, I focus on the rbf kernel. The accuracy score of the rbf kernel classifier is as follows.

Table.3 Accuracy scores of the SVM classifier with different hyperparameter combinations

Model Accuracy	gamma=0.01	gamma=0.1	gamma=1	gamma=10	gamma=100	gamma=1000
C= 0.1	0.371	0.607	0.849	0.929	0.90825	0.422
C=1	0.607	0.84325	0.92125	0.9605	0.969	0.827
C=10	0.84175	0.91525	0.946	0.97275	0.9715	0.8435
C=100	0.9125	0.935	0.962	0.97525	0.97125	0.8435
C=1000	0.93	0.94525	0.96875	0.9685	0.97075	0.8435
C=10000	0.9375	0.96	0.9695	0.9685	0.97075	0.8435

From Table. 3, we can observe an optimal hyperparameter combination, which is kernel = rbf, gamma = 10, C = 100. In the next step, I select a narrower range of the C and gamma around their previous optimal parameters. I set the range of C as 10-100 and the range of gamma as 2-20. I interpret the results of the second GridSearchCV in the format of the heat map. The code is the following:

```
# Heat Map
scores = grid_gamma_results.cv_results_['mean_test_score'].reshape(10,10)
param_C = [10,20,30,40,50,60,70,80,90,100]
param_gamma = [2,4,6,8,10,12,14,16,18,20]
plt.figure(figsize=(10, 10))
plt.subplots_adjust(left=.2, right=0.95, bottom=0.35, top=0.75)
plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot)
plt.ylabel('param_C')
plt.xlabel('param_gamma')
plt.colorbar()
plt.yticks(np.arange(len(param_C)), param_C)
plt.xticks(np.arange(len(param_gamma)), param_gamma)
plt.title('Grid Search Accuracy Score')
plt.show()
```

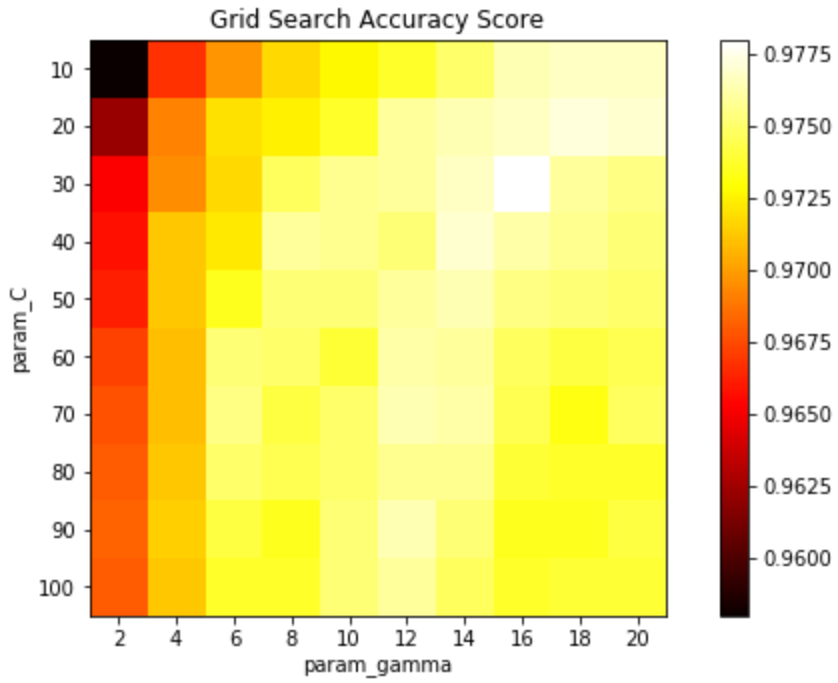

Fig.1 Accuracy scores of the SVC with different hyperparameter combination

From Fig.1, we can observe that kernel = rbf, gamma= 16, C= 30 is the optimal hyperparameter combination.

4. Model training and testng

I choose the optimal hyperparameter (kernel = rbf, gamma= 16, C= 30) and train the support vector classifier model with the 40000 training data and save the well-trained model as a binary file s.t. Model.dat. The code is as follows:

```
from sklearn.svm import SVC
clf = SVC(kernel='rbf', C= 30, gamma= 16)
clf.fit(X,y)
import pickle
pickle.dump(clf, open("model.dat", "wb"))
```

The information of my classifier is as follows:

```
SVC(C=30, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=16, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Next I read the support vector classifier from model.dat by running satellite.py under the 100,000 test data. Finally I create an ASCII file landuse.csv that consists of a single line with the 100,000 classification decisions (represented as the ASCII strings ’barren land’, ’trees’, ’grassland’, and ’none’) separated by commas. The output single line csv table and coding are as follows.

Table. 4 The output CSV file

barren land	barren land	none	grassland	grassland	trees	barren land	barren land	none	barren land

```
import pickle
import numpy as np
loaded_model = pickle.load(open("model.dat", "rb"))
testy=loaded_model.predict(test_x)

r=map(int,testy)
L = ['barren land', 'trees', 'grassland', 'none']
s = ','.join([L[t] for t in r])
f = open('landuse.csv', 'w')
f.write(s)
f.close()
```

At last, I test the well-trained support vector classifier under the 100,000 test dataset. I use the sklearn.metric module to evaluate the model performance. The result shows the model achieve an accuracy of 98.8% to classify various land cover classes.

```
# prediction accuracy
y_pred = clf.predict(test_x)
y_true = test_y
from sklearn.metrics import accuracy_score
accuracy_score(y_true, y_pred)
```

5. Data visualization

In order to visualize the classification results, I use the matplotlib.pyplot and seaborn to plot the heatmap.

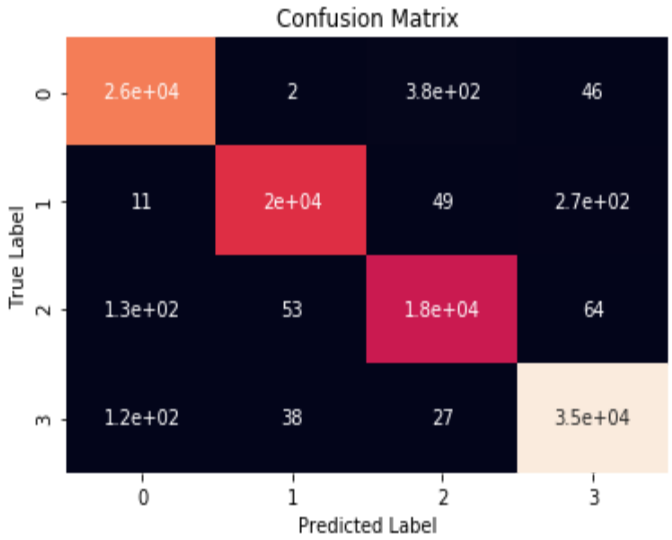

Fig. 2 the heat map of the support vector classification results

```
# confusion matrix
from sklearn.metrics import confusion_matrix
```

```
matrix = confusion_matrix(y_true, y_pred)

# creating heatmap
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.heatmap(matrix,annot=True,cbar=False)
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix')
```

6 random forest

The workflow of the random forest is similar to the support vector machine. The different things which we need to pay attention to is the hyperparameter tuning.

In the random forest, important hyperparameters like n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf are considered. GridSearchCV is used under the range of max_depth between 200-400, range of max_features between 2-6, range of min_samples_leaf between 2-4, range of min_samples_split between 2-4, range of n_estimators between 1000-2000.

The code of the parameter tuning is as follows.

```
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search
param_grid = {
   'bootstrap': [False],
   'max_depth': [200, 300, 400],
   'max_features': [2, 4, 6],
   'min_samples_leaf': [2, 3, 4],
   'min_samples_split': [ 2, 3, 4],
   'n_estimators': [1000,1500,2000]
}

from sklearn.ensemble import RandomForestClassifier
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2, scoring='accuracy')
# Fit the grid search to the data
grid_search.fit(X, y)
grid_search.best_params_
```

The information of my random forest classifier is as follows:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=300, max_features=2, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
```

```
min_samples_leaf=2, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=1500, n_jobs=None,
oob_score=False, random_state=None, verbose=0,
warm_start=False)
```

I also test the well-trained random forest classifier under the 100,000 test dataset. I use the sklearn.metric module to evaluate the model performance. The result shows the model achieve an accuracy of 98.372% to classify various land cover classes. The classification result is transform into the heat map for data visualization.

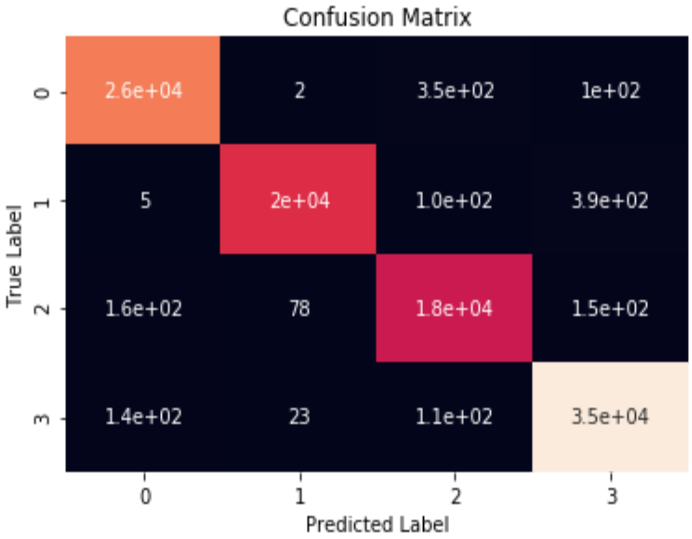

Fig. 3 the heat map of the random forest classification results

Acknowledged from Professor Wagner, TA, cx223

Tian Yu ty364
2019.12.9