

Exploration of Cover Types

Capstone Project_Week 2

Lisha Yao - August 28, 2020

Outline

- Introduction
- Data Describing
- Methodology
- Result/Conclusion

Introduction

- Environmental issues have gradually come into the public eye.
- About 690 million people are going hungry in 2019, 10 million more than in 2018, and nearly 60 million more than in the past five years.
- Both of these aspects undoubtedly show the important role of planting in the operation of the world environment.

Introduction

- A data set of nearly 500 thousands type of different covers in the world is used in this project.
- The result could be wildly used both in agricultural corporation or environmental corporation.
- By considering the output of the data set, the person in agricultural and environmental filed would be interested.

Data Describing

- Source: <https://www.kaggle.com/abdullahahmedattahir/covtype>

```
In [6]: df.shape
Out[6]: (581012, 55)

In [7]: df.head()
Out[7]:
   Elevation  Aspect  Slope  Horizontal_Distance_To_Hydrology  Vertical_Distance_To_Hydrology  Horizontal_Distance_To_Roadways  Hillshade_9am  Hillshade_Noon
0        2596      51       3                           258                               0                           510                  221                  232
1        2590      56       2                           212                              -6                           390                  220                  235
2        2804     139       9                           268                               65                          3180                  234                  238
3        2785     155      18                           242                             118                          3090                  238                  238
4        2595      45       2                           153                              -1                           391                  220                  234
5 rows × 55 columns
```

Figure 1. Raw data

Data Describing

```
cover_type_counts = df['Cover_Type'].value_counts().rename_axis('cover_type').to_frame('number_counts')
cover_type_counts = cover_type_counts.sort_index(ascending = True)
#cover_type_counts.columns = 'number_counts'
#cover_type_counts.index.name = 'cover_type'

cover_type_counts
```

cover_type	number_counts
1	211840
2	283301
3	35754
4	2747
5	9493
6	17367
7	20510

Figure 2. Count by types

```
#cover_type_counts = cover_type_counts.transpose()
cover_type_counts.plot(kind='bar')

plt.title('Histogramme of the number counting for each cover type')
plt.xlabel('Cover Type')
plt.ylabel('Number Counting')
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```

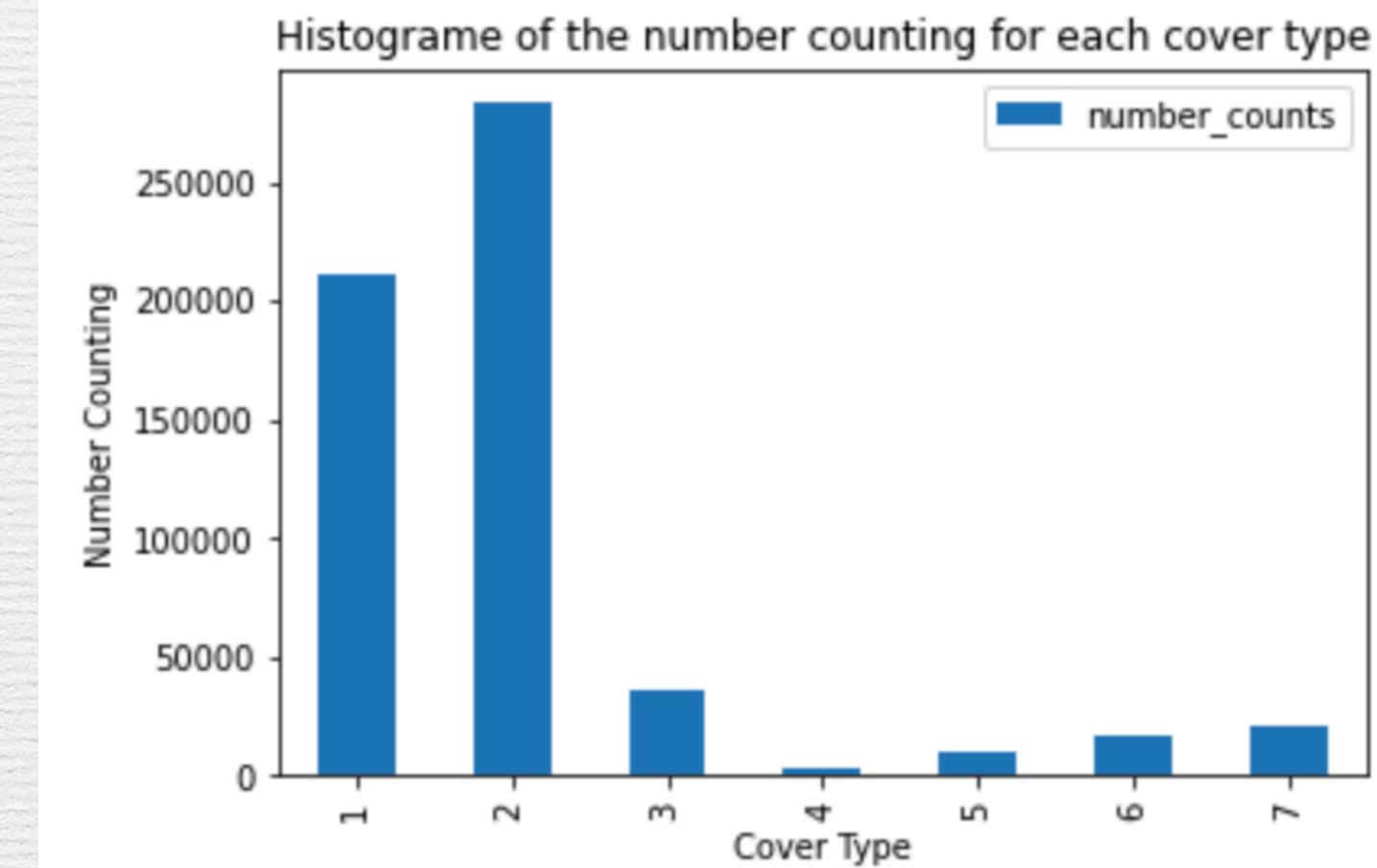


Figure 3. Histogram of cover types

Methodology - Sampling

- This dataset is highly skewed and thus we need to sample from each category to construct the new dataset for both training and testing purpose.
- We'll run the sampling policy 5 times and use the average score as the final result.

```
d=[]
# in each category, we sample 2000 samples
num_samples=2000
for i in range(n_class):
    df_sub=data[data['Cover_Type']==i+1]
    # sample this subset
    df_sub=df_sub.sample(num_samples)
    # reset the index
    df_sub=df_sub.reset_index(drop=True,inplace=False)
    # append it into list
    d.append(df_sub)

# concat sublist together
d_init=d[0]
for i in range(1,n_class):
    # update the initial dataset
    d_init=pd.concat(objs=[d_init,d[i]],axis=0)
    # reset index
    d_init=d_init.reset_index(drop=True,inplace=False)

# update the dataset
d=d_init
print('The sampling from original dataset is finished')
```

The sampling from original dataset is finished

Figure 4. Sampling

Methodology - Dataset

- The next thing is to separate the dataset into training and test set. We'll run the sampling policy 5 times and use the average score as the final result.
- 80% of the training set and 20% of the test set from the original dataset.

```
: # split the dataset into input X and outcome Y
X=d.iloc[:, :-1]
Y=d.iloc[:, -1]
# apply one-hot encoding on Y
y=Y.values
Y=np.zeros((len(y),n_class))
for i in range(Y.shape[0]):
    ind=int(y[i])
    Y[i,ind-1]=1

# apply the train test split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,shuffle=True)

print('The shape of training set input is: '+str(X_train.shape))
print('The shape of training set outcome is: '+str(Y_train.shape))
print('The shape of test set input is: '+str(X_test.shape))
print('The shape of test set outcome is: '+str(Y_test.shape))
```

The shape of training set input is: (11200, 54)
The shape of training set outcome is: (11200, 7)
The shape of test set input is: (2800, 54)
The shape of test set outcome is: (2800, 7)

Figure 5. Dataset

Methodology - Classification Algorithm

- After this, I implement three different types of classification algorithms:

KNN

SVM

Logistic
regression

Methodology - KNN

```
# apply knn algorithm
from sklearn.neighbors import KNeighborsClassifier

# try with 10 different values of k to find the best one
Ks=10
mean_acc=np.zeros((Ks-1))
std_acc=np.zeros((Ks-1))
CM=[];
mean_acc=np.zeros((Ks,))
for n in range(1,Ks+1):
    #Train Model and Predict
    neigh=KNeighborsClassifier(n_neighbors=n).fit(X_train,Y_train)
    y_hat=neigh.predict(X_test)
    # get the accuracy rate
    y_pred=np.argmax(y_hat,axis=1)
    y_true=np.argmax(Y_test,axis=1)
    acc=np.sum((y_true==y_pred)*1)/Y_test.shape[0]
    mean_acc[n-1]=acc
    # get the confusion matrix
    d=cm(y_true,y_pred)
    CM.append(d)
```

Figure 6. KNN

Methodology - SVM

```
# apply SVM algorithm
from sklearn import svm
clf=svm.LinearSVC(penalty='l1',dual=False)
clf.fit(X_train,np.argmax(Y_train,axis=1))
y_hat=clf.predict(X_test)

# get the accuracy rate
y_pred=y_hat
y_true=np.argmax(Y_test,axis=1)
acc=np.sum((y_true==y_pred)*1)/Y_test.shape[0]
# get the confusion matrix
d=cm(y_true,y_pred)
```

Figure 7. SVM

Methodology - Logistic Regression

```
# apply logistic regression
from sklearn.linear_model import LogisticRegression
LogR=LogisticRegression(penalty='l2',C=0.01,solver='lbfgs',multi_class='multinomial').fit(X_train,np.argmax(Y_train, axis=1))
y_hat=LogR.predict(X_test)

# get the accuracy rate
y_pred=y_hat
y_true=np.argmax(Y_test, axis=1)
acc=np.sum((y_true==y_pred)*1)/Y_test.shape[0]
# get the confusion matrix
d=cm(y_true,y_pred)
```

Figure 8. Logistic Regression

Result / Conclusion

```
# test result
print('The best number of neighbor is: '+str(np.argmax(mean_acc)+1))
print('And the correspond accuracy rate is: '+str(mean_acc[np.argmax(mean_acc)]))
print()
print('The best confusion matrix is: ')
print(CM[np.argmax(mean_acc)])
```

The best number of neighbor is: 1
And the correspond accuracy rate is: 0.8167857142857143

Figure 8. KNN result

```
# test result
print('The test accuracy is: '+str(acc))
print()
print('The confusion matrix is: ')
print(d)
```

The test accuracy is: 0.6778571428571428

Figure 9. SVM result

```
# test result
print('The test accuracy is: '+str(acc))
print()
print('The confusion matrix is: ')
print(d)
```

The test accuracy is: 0.6075

Figure 10. Logistic Regression result

After comparing the three different type of classification algorithm, as a result, KNN has the best result.