LAMPIRAN

Lampiran 1. Source Code Maximum Likelihood Clasifier

```
import numpy as np

class MLClassifier:
    def fit (self, x: np.ndarray, y: np.ndarray) -> None:
        # no. of variables / dimension
        self.d = x.shape[1]

        # no. of classes;
        self.nclasses = len(set(y))

        # list of means; mu_list[i] is mean vector for label i
        self.mu_list = []

        # list of inverse covariance matrices;
        # sigma_list[i] is inverse covariance matrix of label i
        # for efficiency reasons we store only the inverses
        self.sigma_inv_list = []

        # list of scalars in front of e^...
        self.scalars = []
        n = x.shape[0]
        for i in range(self.nclasses):
            # subset of obesrvations for label i
            cls_x = np.array([x[j] for j in range(n) if y[j] ==
i+1])

            mu = np.mean(cls_x, axis=0)

            sigma = np.cov(cls_x, rowvar=False)
            if np.sum(np.linalg.eigvals(sigma) <= 0) != 0:
                print(f'{cls_x} is not postive definite!')

            sigma_inv = np.linalg.inv(sigma)

            scalar = 1/np.sqrt(((2*np.pi**self.d)*np.linalg.det
(sigma))

            self.mu_list.append(mu)
            self.sigma_inv_list.append(sigma_inv)
            self.scalars.append(scalar)
```

```
def _class_likelihood(
      self, x: np.ndarray,
      cls: int
) -> float:
        '''
        x - numpy array of shape (d,)
        cls - class label

        Returns: likelihood of x
           under the assumption that class label is cls
        '''
        mu = self.mu_list[cls]
        sigma_inv = self.sigma_inv_list[cls]
        scalar = self.scalars[cls]
        d = self.d

        exp = (-1/2*np.dot(np.matmul(x-mu, sigma_inv), x-mu)

        return scalar * (np.e**exp)

def predict (self, x: np.ndarray):
        '''
        x - numpy array of shape (d,)
        Returns: predicted label
        '''
        likelihoods = [self.class_likelihood(
            x, i) for i in range(self.nclasses)]
        return np.argmax(likelihoods)

def score (self, x: np.ndarray, y: np.ndarray) -> float:
        '''
        x - numpy array of shape (n, d); n = #observations;
           d = #variables
        y - numpy array of shape (n,)
        Returns: accuracy of predictions
        '''
        n = x.shape[0]
        predicted_y = np.array([self.predict(x[i]) for i in
range(n)])
        n_correct = np.sum(predicted_y+1 == y)
        return n_correct/n
```

Lampiran 2. Source Code Model Support Vector Machine

```
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

clf = SVC(kernel="rbf")

scores = []

foldke = 1
for train_index, test_index in kf.split(X):
    print(f"fold ke-{foldke}")
    X_train, X_test = X.iloc[train_index],
X.iloc[test_index]
    y_train, y_test = y.iloc[train_index],
y.iloc[test_index]

    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    score = clf.score(X_test, y_test)
    scores.append(score)

    cm = confusion_matrix(y_test, y_pred)
    print(f"Confusion Matrix:\n{cm}\n")

    report = classification_report(y_test, y_pred)
    print(f"Classification Report:\n{report}\n")

    model_output_file = f'./model/svm_rbf_fold_{foldke}'
    joblib.dump(clf, f'{model_output_file}.joblib')
    foldke += 1

print(f"Average Score = {np.average(scores)}")
```

Lampiran 3. Source Code Model Random Forest

```
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

clf = RandomForestClassifier()

scores = []

foldke = 1
for train_index, test_index in kf.split(X):
    print(f"fold ke-{foldke}")
    X_train, X_test = X.iloc[train_index],
X.iloc[test_index]
    y_train, y_test = y.iloc[train_index],
y.iloc[test_index]

    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    score = clf.score(X_test, y_test)
    scores.append(score)

    cm = confusion_matrix(y_test, y_pred)
    print(f"Confusion Matrix:\n{cm}\n")

    report = classification_report(y_test, y_pred)
    print(f"Classification Report:\n{report}\n")

    model_output_file = f'./model/rf_fold_{foldke}'
    joblib.dump(clf, f'{model_output_file}.joblib')
    foldke += 1

print(f"Average Score: {np.average(scores)}")
```

Lampiran 4. Source Code Model Maximum Likelihood

```
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

clf = MLClassifier()

scores = []
conf_m = []

foldke = 1
for train_index, test_index in kf.split(X):
    print(f"fold ke-{foldke}")
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    clf.fit(X_train, y_train)

    y_pred = np.array([clf.predict(i) + 1 for i in X_test])

    score = clf.score(X_test, y_test)
    scores.append(score)

    cm = confusion_matrix(y_test, y_pred)
    conf_m.append(cm)
    print(f"confusion matrix\n {cm}\n")

    report = classification_report(y_test, y_pred)
    print(f"Classification Report:\n{report}\n")

    model_output_file = f'./model/mlc_fold_{foldke}'
    joblib.dump(clf, f'{model_output_file}.joblib')
    foldke += 1

print(f"Average Score: {np.average(scores)}")
```

Lampiran 5. Source Code Cloud Masking di GEE

```
// Create a Sentinel-2 Level-2A image collection
var sentinel2Collection = s2
  .filterBounds(roi)
  .filterDate(startDate, endDate)
  .filterMetadata('CLOUDY_PIXEL_PERCENTAGE', 'less_than',
10);

// Function to mask based on QA band
var maskBasedOnQA = function(image) {
  // Select the QA60 band
  var QA60 = image.select(['QA60']);
  var mask = QA60.eq(0);
  // Update the image mask
  return image.updateMask(mask);
};
```