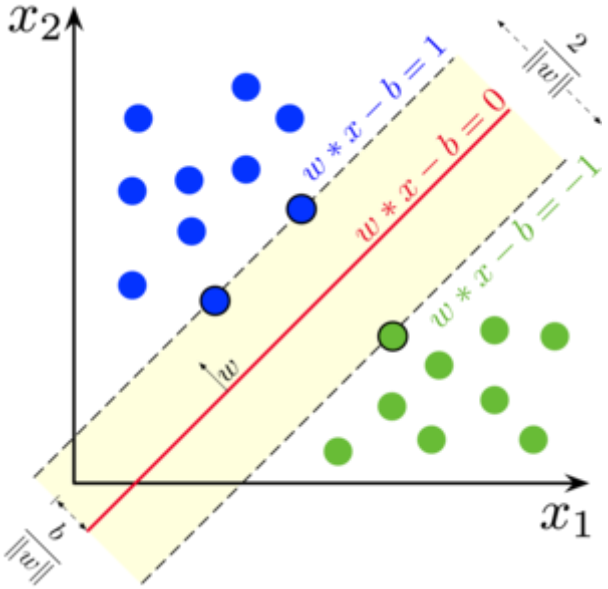
نام و نام خانوادگی: سروش مهربان
شماره دانشجویی: 9628099

تمرین عملی سوم داده کاوی

۱-

الف) توضیح ریاضیات کد:



همانطور که میدانیم در SVM هدف ماکسیمم کردن $\frac{2}{||w||}$ می‌باشد. مطابق با این ویدیو، میتوان از Hinge loss و گرادیان برای حل مساله SVM استفاده کرد. به طوری که داریم:

$$Hinge\ loss: l = max\big(0,1 - y_i(w \cdot x_i - b)\big)$$

طبق رابطه Hinge loss، زمانی که داده ها به درستی طبقه بندی میشوند نتیجه صفر و در غیر این صورت $1 - y_i(w \cdot x_i - b)$ می‌باشد. پس مینیمم زمانی رخ میدهد که طبقه‌بندی تمام نقاط به درستی صورت گرفته که هدف مساله ما می‌باشد. اما از طرفی هدف ماکسیمم کردن margin یا مینیمم کردن معکوس آن نیز می‌باشد. لذا با اضافه کردن Regularization داریم:

$$Cost\ Function: J\ = \lambda||w||^2 + \frac{1}{n}\sum_{i=1}^{n} max\big(0,1 - y_i(w \cdot x_i - b)\big)$$

با توجه به تابع هزینه فوق، میتوان گفت زمانی که داده درست طبقه بندی میشود یا به عبارت دیگر اگر داشته باشیم $1 - y_i(w \cdot x_i - b)\ \geq\ 1$ آنگاه می‌توان گفت تابع هزینه مربوط به آن داده برابر است با:

$$J_i\ = \lambda||w||^2$$

و درصورت طبقه بندی اشتباه نیز برابر است با:

$$J_i = \lambda||w||^2 + max(0,1 - y_i(w \cdot x_i - b))$$

برای حل بایستی یکبار نسبت به w_k و بار دیگر نسبت به b مشتق جزئی بگیریم و داریم:

$if\ 1 - y_i(w \cdot x_i - b) \geq 1$:

$$\frac{\partial J_i}{\partial w_k} = 2\lambda w_k$$

$$\frac{\partial J_i}{\partial b} = 0$$

else:

$$\frac{\partial J_i}{\partial w_k} = 2\lambda w_k - y_i x_i$$

$$\frac{\partial J_i}{\partial b} = y_i$$

سپس طبق رابطه گرادیان میتوان نوشت:

$$w_{new} = w_{old} - \alpha dw$$

$$b_{new} = b_{old} - \alpha db$$

حال به توضیح کد میپردازیم:

```
class LinearSVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iter=1000):
        """
        LinearSVM using hinge loss and gradient descent

        :param learning_rate: Learning rate of gradient descent optimization
        :param lambda_param: Regularization Parameter
        :param n_iter: Number of iterations
        """
        self.learning_rate = learning_rate
        self.lambda_param = lambda_param
        self.n_iter = n_iter

        self.w = None
        self.b = None
```

در LinearSVM، ابتدا Hyperparameter ها تحت ورودی دریافت و در فیلد کلاس قرار میگیرند. لازم به ذکر است که پارامتر های w و b از ابرصفحه در ابتدا None می‌باشند که در ادامه پس از یادگیری مدل مقداگیری میکنند.

```
def train(self, X, y):
    """
    Trains the Linear SVM model using gradient descent

    :param X: Training features
    :param y: Training labels
    """
    n_features = X.shape[1]

    """
    Since naturally SVM classifies our data as 1 and -1 and given dataset's
labels are 0 and 1, we use the following
    function to convert 0 labels to -1 and others to 1
    """
    y = np.where(y == 0, -1, 1)

    self.w = np.zeros(n_features)
    self.b = 0

    for iteration in range(self.n_iter):
        for i, x_i in enumerate(X):
            classifies_correctly = y[i] * (np.dot(x_i, self.w) - self.b) >= 1
            if classifies_correctly:
                self.w -= self.learning_rate * (2 * self.lambda_param *
self.w)
            else:
                self.w -= self.learning_rate * (2 * self.lambda_param *
self.w - np.dot(x_i, y[i]))
                self.b -= self.learning_rate * y[i]
```

در تابع train، هدف بدست آوردن پارامتر های w و b به کمک روش گرادیان می‌باشد. به طوری که در ابتدا مقادیر متغیر های مستقل(X) و وابسته(y) از داده آموزشی دریافت میشوند.سپس با توجه به این که SVM داده هارا به 1- و 1 طبقه بندی میکند، به کمک np.where کلاس های صفر دیتاست به 1- و باقی به 1 نگاشت میشوند. در ادامه در یک حلقه به ازای تک تک داده های آموزشی، ابتدا شرط این که به کمک پارامتر های فعلی w و b درست طبقه بندی میشوند یا خیر بررسی می‌شود. طبق روابط توضیح داده شده در صورت طبقه بندی درست یا نادرست با توجه به وجود max در تابع هزینه، گرادیان به نحوه مختلف حساب میشود. پس در یک شرط بررسی میشود و در صورت طبقه بندی درست و یا نادرست، به روش مربوطه، گرادیان محاسبه شده و پارامتر های w و b بروز میشوند.

```
def predict(self, X):
    """
    Classifies given data as 1 and -1

    :param X: given data
    :return: Labels of each data as 1 and -1
    """
    try:
        return np.sign(np.dot(X, self.w) - self.b)
    except TypeError:
        print("ERROR: You have to train the model first")
        exit()
```

در تابع Predict نیز هدف طبقه بندی داده های ورودی می‌باشد. بدین صورت که رابطه $w^T x - b$ محاسبه می‌شود و اگر بزرگتر تر از صفر باشد به 1 و در غیر این صورت به 1- نگاشت میشوند. لذا از تابع sign برای این‌کار استفاده شده است.

لازم به ذکر است که قبل از فراخوانی این متد بایستی متد train فراخوانی شود. برای اطمینان از این امر در هنگام ساخت شی از این کلاس مقدار پارامتر های w و b به صورت None قرار گرفته شده اند. پس هنگام فراخوانی این متد و هنگام محاسبه np.dot با توجه به None بودن w، TypeError Exception رخ خواهد داد و به کاربر خطای مربوطه را نمایش می‌دهیم.

```
@staticmethod
def calculate_accuracy(y_true, y_pred):
    """
    Calculates accuracy of our model

    :param y_true: Our labels
    :param y_pred: Model prediction
    :return: Accuracy fo our model
    """
    y_true = np.where(y_true == 0, -1, 1)

    correct_classifications = np.count_nonzero(y_true == y_pred)
    n_samples = y_true.shape[0]

    return (correct_classifications / n_samples) * 100
```

مشابه تمرین گذشته تابع calculate_accuracy نیز موجود می‌باشد. در این تابع y_true == y_pred شده و حاصل آن یک آرایه از True و False می باشد که در صورت cast شدن به صفر و یک cast می‌شوند. پس به کمک متد np.count_nonzero میتوان تعداد true هارا شمرد که تعداد طبقه بندی های درست را برمیگرداند. در آخر این مقدار تقسیم بر کل مقدار نمونه شده و در ۱۰۰ ضرب میشود.

```
def visualize(self, X):
    def get_hyperplane_point(x, w, b, offset):
        """
        Finds a point on the following hyperplane:
        w0x + w1y - b = offset => y = (offset - w0x + b) / w1

        :param x: independent variable of the hyperplane
        :param w: weights of hyperplane
        :param b: y-intercept
        :param offset: Either 1, 0, or -1
        :return: A point on the hyperplane
        """
        y = (offset - w[0] * x + b) / w[1]
        return y

    y_pred = self.predict(X)

    fig, ax = plt.subplots()
    ax.scatter(x=X[:, 0], y=X[:, 1], c=y_pred)

    min_x0 = np.amin(X[:, 0])
    max_x0 = np.amax(X[:, 0])

    x1_1_1 = get_hyperplane_point(min_x0, self.w, self.b, offset=-1)
    x1_1_2 = get_hyperplane_point(max_x0, self.w, self.b, offset=-1)

    x1_mid_1 = get_hyperplane_point(min_x0, self.w, self.b, offset=0)
    x1_mid_2 = get_hyperplane_point(max_x0, self.w, self.b, offset=0)

    x1_2_1 = get_hyperplane_point(min_x0, self.w, self.b, offset=1)
    x1_2_2 = get_hyperplane_point(max_x0, self.w, self.b, offset=1)

    ax.plot([min_x0, max_x0], [x1_1_1, x1_1_2], 'k')
    ax.plot([min_x0, max_x0], [x1_mid_1, x1_mid_2], 'y--')
    ax.plot([min_x0, max_x0], [x1_2_1, x1_2_2], 'k')

    plt.show()
```

در متد Visualize نیز هدف رسم نمودار می‌باشد. برای این کار به کمک تابع get_hyperplane_plot مختصات نقاط مربوط به ۳ خط جدا کننده را بدست می‌آوریم. به طوری که همانطور که در کامنت توضیح داده‌شده، با ورودی x نقطه، y نقطه بدست می آید که با تغییر Offset بین مقادیر 1 ,0 ,1-، سه خط SVM را میتوانیم تشکیل دهیم.

حال تنها کافیست از هر خط تنها ۲ نقطه مینیمم و ماکسیمم را بدست آوریم و با وصل کردن آن ها خط را تشکیل دهیم. پس به کمک np.amin و np.amax مقادیرx0 مینیمم و ماکسیمم را بدست می آوریم و به کمک تابع get_hyperplane_point نیز مقادیر x1 متناظر با این نقاط در ۳ خط را بدست می آوریم و به کمک ax.plot نیز خطوط را رسم میکنیم.

SVM

Data1

```
In [2]: data1 = pd.read_csv('data/data1.csv')
```

```
In [3]: # Extract features and labels
        X = data1.drop('Class', axis=1).to_numpy()
        y = data1['Class'].to_numpy()
```

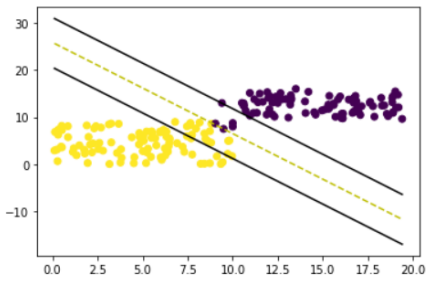
```
In [4]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, shuffle=False)
```

```
In [5]: classifier = LinearSVM()

        classifier.train(X_train, y_train)

        y_pred_train = classifier.predict(X_train)
        y_pred_test = classifier.predict(X_test)

        train_accuracy = classifier.calculate_accuracy(y_train, y_pred_train)
        test_accuracy = classifier.calculate_accuracy(y_test, y_pred_test)

        print(f"Model accuracy:\n"
              f"Training data: {train_accuracy:.2f}%\n"
              f"Test data: {test_accuracy:.2f}%")
```

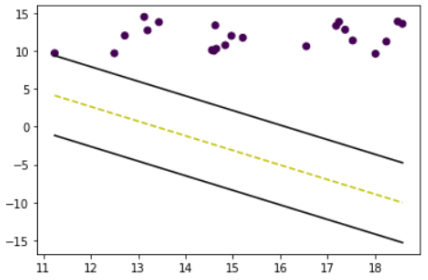
```
Model accuracy:
Training data: 98.42%
Test data: 100.00%
```

در فایل ژوپیتر نوتبوک نیز مشاهده میکنیم که مقدار دقت در داده های آموزشی و تست در data1 به ترتیب 98.42% و 100% می‌باشد.

```
In [6]: classifier.visualize(X_train)
```




```
In [7]: classifier.visualize(X_test)
```



نمودار های خواسته شده نیز در تصویر فوق مشاهده میکنیم.

Data2

In [8]:
```
data2 = pd.read_csv('data/data2.csv')
```

In [9]:
```
# Extract features and labels
X = data2.drop('Class', axis=1).to_numpy()
y = data2['Class'].to_numpy()
```

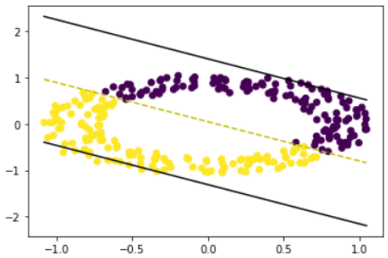
In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, shuffle=False)
```

In [11]:
```
classifier = LinearSVM()

classifier.train(X_train, y_train)

y_pred_train = classifier.predict(X_train)
y_pred_test = classifier.predict(X_test)

train_accuracy = classifier.calculate_accuracy(y_train, y_pred_train)
test_accuracy = classifier.calculate_accuracy(y_test, y_pred_test)

print(f"Model accuracy:\n"
      f"Training data: {train_accuracy:.2f}%\n"
      f"Test data: {test_accuracy:.2f}%")
```

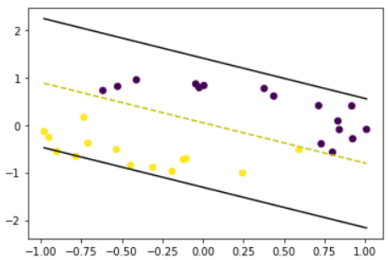
```
Model accuracy:
Training data: 50.74%
Test data: 43.33%
```

اما هنگامی که میخواهیم از data2 استفاده کنیم، مشاهده میکنیم دقت به مراتب کمتر و به ترتیب برای داده‌های آموزشی و تست برابر با 50.74% و 43.33% می‌باشد.

In [12]:
```
classifier.visualize(X_train)
```



In [13]:
```
classifier.visualize(X_test)
```



با رسم شکل نیز مشاهده میکنیم که داده ها یک توزیع دونات شکل دارند و با یک خط نمیتوان آن هارا جدا کرد. لذا Linear SVM عملکرد خوبی را برای این داده ها ندارد.

Non-linear SVM

In [14]:
```
classifier = SVC(kernel='rbf')

classifier.fit(X_train, y_train)

train_accuracy = classifier.score(X_train, y_train)
test_accuracy = classifier.score(X_test, y_test)

print(f"Model accuracy:\n"
      f"Training data: {train_accuracy * 100 :.2f}%\n"
      f"Test data: {test_accuracy * 100 :.2f}%")
```

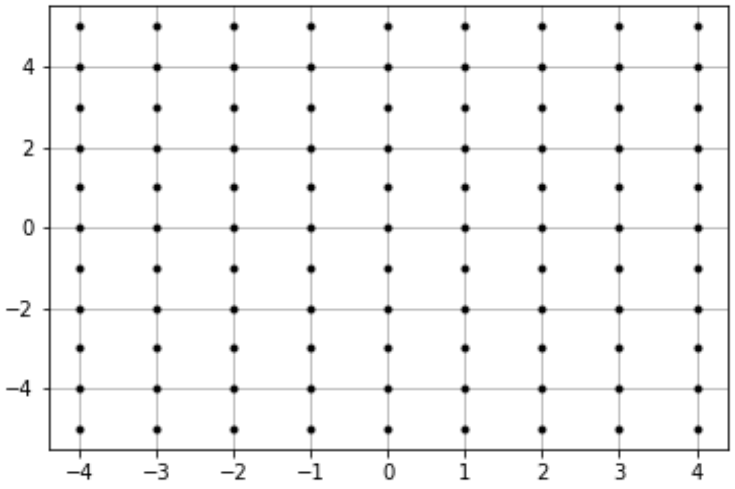
```
Model accuracy:
Training data: 98.15%
Test data: 100.00%
```

اما هنگام استاده از Non-linear SVM به کمک کتابخانه sklearn و استفاده از RBF برای kernel function، مشاهد همیکنیم دقت به 98.15% و 100% خواهد رسید.

برای رسم Decision Boundary نیاز است که از کانتور استفاده کنیم. برای نحوه رسم آن از stackoverflow کمک گرفته شده است. در ابتدا به بررسی توابع مربوطه میپردازیم:

In [15]:
```
# From https://stackoverflow.com/questions/51297423/plot-scikit-learn-sklearn-svm-decision-boundary-surface

def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    return out
```

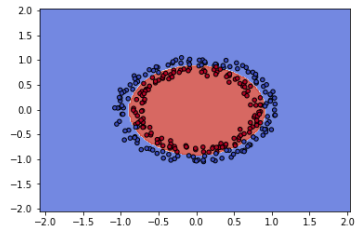
در تابع make_meshgrid، محور X و Y داده ها تحت ورودی داده میشود. سپس مینیمم و ماکسیمم هریک گرفته شده و به کمک تابع np.meshgrid، ماتریس مختصات همانند شکل زیر حاصل میشود:



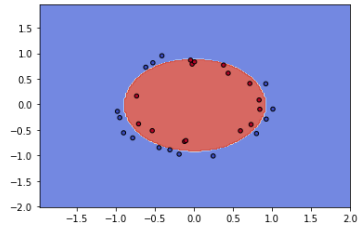
در این ماتریس مختصات، تمام نقاط دایره ای شکل یک درایه از ماتریس meshgrid می‌باشد. با این تفاوت که در کد فوق، فاصله بین دایره ها 0.2 و در شکل فوق ۱ می‌باشد. پس ابتدا روی تمامی فضای مختصات، نقاطی با فاصله یکسان تولید میکنیم.

سپس در تابع plot_contours به ازای تک تک این نقاط، کلاس مربوطه محاسبه شده و در متغیر Z قرار میگیرد. سپس به کمک countourf، نمودار کانتور رسم میشود و نشان داده میشود که در چه نواحی چه کلاسی اختیار میگیرد. برای رنگ بندی کانتور از plt.cm.coolwarm استفاده میشود که رنگ گرم و سرد (قرمز و آبی) باشد و برای میزان شفافیت نیز از مقدار alpha=0.8 استفاده شده تا کمی کمرنگ شوند.

```
xx, yy = make_meshgrid(X_train[:, 0], X_train[:, 1])

plot_contours(ax, classifier, xx, yy)
ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors='k')

plt.show()
```



```
In [17]: fig, ax = plt.subplots()

xx, yy = make_meshgrid(X_test[:, 0], X_test[:, 1])

plot_contours(ax, classifier, xx, yy)
ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=plt.cm.coolwarm, s=20, edgecolors='k')

plt.show()
```



همانطور که در شکل فوق میبینیم، نواحی کانتور مشخص شده اند و نقاط قرمز و آبی نیز به کمک scatter رسم شده اند.

۲- پیاده سازی زیر از این ویدیو یوتیوب کمک گرفته شده است. هرچند تغییراتی در پیاده سازی وجود دارد که در زیر به توضیح آن پرداخته شده است.

برای پیاده‌سازیRandom Forest از کد Decision Tree تمرین قبل استفاده می‌کنیم. با این تفاوت که در تمرین گذشته ستون های Categorized بوده و هنگام ساختن Node فرزند دسته صفر به یک سمت و یک به سمت دیگر میرفتند. اما در حال حاضر داریم:

```
@staticmethod
def binary_split(feature_index, value, data):
    left = data[data[:, feature_index] < value]
    right = data[data[:, feature_index] >= value]
    return np.array([left, right], dtype=object)
```

همانطور که مشاهده میشود، یک پارامتر value اضافه شده است و در ستون مورد نظر مواردی که از value کمتر هستند به یک فرزند و باقی به سمت دیگر میروند.

```
def get_split(self, data):
    best_index, best_value, best_score, best_groups = float('inf'),
float('inf'), float('inf'), None
    for feature_index in range(data.shape[1] - 1):  # except last column
        for row in data:
            groups = self.binary_split(feature_index, row[feature_index],
data)
            cost_score = self.cost(groups)
            if cost_score < best_score:
                best_index, best_value, best_score, best_groups =
feature_index, row[feature_index], cost_score, \
                                                                    groups
    return {
        'index': best_index,
        'value': best_value,
        'groups': best_groups
    }
```

در این متد نیز مشاهده میکنیم که به ازای تمام درایه های موجود در ستون، binary_split محاسبه میشود، پس به ازای تمامی درایه های ستون ۲ فرزند تشکیل میدهیم و برای هر کدام تابع هزینه (که GINI به طور پیشفرض است) محاسبه میشود و آن که از همه بهتر است انتخاب میشود.

حال به پیاده سازی Random Forest میپردازیم:

```
class RandomForest:
    def __init__(self, train_set, n_trees=100, max_depth=3, min_size=10,
criterion='GINI', seed=None):
        """
        :param train_set: train set of our dataset which is a numpy array
        :param n_trees: Number of random trees used in random forest
        :param max_depth: maximum depth that we want to expand. Default is 3
        :param min_size: minimum size of each node. Default is 10
        :param criterion: Either GINI or ENTROPY. Default is GINI
        :param seed: Initial random seed
        """
        self.train_set = train_set
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.min_size = min_size
        self.criterion = criterion

        self.trees = []
        if seed is not None:
            np.random.seed(seed)
        self.train()
```

در هنگام ساخت یک شی از کلاس RandomForest، علاوه بر n_trees که تعداد درخت های تصادفی را نشان میدهد، باقی پارامتر ها برای پارامتر درخت تصمیم میباشند. پس همگی فیلد ها مقدار اولیه میگیرند و در صورت seed تعیین شده، آن مقدار قرار میگیرد تا در اجرا های مختلف نتیجه مختلف نداشته باشیم. سپس متد train فراخوانی میشود.

```
def bootstrap_sample(self):
    """
    Selects random samples from train set with the same size and with
replacement.
    """
    n_samples = self.train_set.shape[0]
    random_indices = np.random.choice(n_samples, size=n_samples,
replace=True)
    return self.train_set[random_indices]

def train(self):
    """
    Trains random forest by creating `n_trees` number of trees with different
bootstrap samples and training them.
    """
    for _ in range(self.n_trees):
        random_train_set = self.bootstrap_sample()
        tree = DecisionTree(random_train_set, self.max_depth, self.min_size,
self.criterion)
        self.trees.append(tree)
```

متد train به ازای تعداد درخت های تصادفی، ابتدا یک bootstrap sample روی داده آموزشی انجام میدهد. به نحوی که به کمک np.random.choice به تعداد درایه های داده آموزشی نمونه گیری با

جایگزینی انجام میدهد. سپس یک درخت با نمونه تصادفی با جایگزینی ساخته میشود و به لیست درختان اضافه میگردد.

```
def predict(self, X):
    """
    Predicts the given data label based on majority voting of different trees

    :param X: given data to predict
    """
    tree_predictions = np.array([tree.predict(X) for tree in self.trees])

    """
    Assume we have 3 trees and X has 2 rows. Then tree_predictions is:
    [[label1, label2]
     [label3, label4]
     [label5, label6]]
    Since we want majority voting of different trees, by calling the
following function tree_predictions will be:
    [[label1, label3, label5]
     [label2, label4, label6]]
    """
    tree_predictions = np.swapaxes(tree_predictions, 0, 1)

    final_predictions = []
    for prediction in tree_predictions:
        prediction = np.array(prediction, dtype='int64')
        final_predictions.append(np.bincount(prediction).argmax())
    return np.array(final_predictions)
```

در تابع predict ابتدا به ازای تک تک درخت های موجود در Random Forest پیشبینی صورت میگیرد. سپس همانطور که در کامنت توضیح داده شده، به کمک np.swapaxes، ماتریس tree_predictions را به گونه ای تغییر میدهیم که در هر سطر نشان دهد درخت های مختلف چه پیشبینی روی داده کرده اند. سپس در یک حلقه به کمک np.bincount مشخص میکنیم که کدام کلاس بیشترین frequency پیشبینی را دارد (خروجی یک آرایه است که index آن کلاس مربوطه و درایه تعداد تکرار آن است). پس با argmax() و bincount به اصطلاح majority voting صورت میگیرد و در آخر پیشبینی را برمیگردانیم.

```
def create_confusion_matrix(self, y_true, y_pred):
    confusion_matrix = []

    labels = np.unique(self.train_set[:, -1])
    for label in labels:
        label_indices = np.where(y_true == label)
        label_predictions = y_pred[label_indices]

        label_predictions_count = []
        for predicted_label in labels:
            label_predictions_count.append(np.count_nonzero(label_predictions
== predicted_label))

        confusion_matrix.append(label_predictions_count)
    return confusion_matrix
```

برای Confusion Matrix نیز ابتدا تمام label های داده آموزشی را در متغیر labels قرار میدهیم. سپس به ازای تک تک label ها ابتدا در بردار لیبل های داده شده (y_true) ایندکس های متناظر label موجودر در حلقه را جدا کرده و در label_indices میگذاریم. سپس در بردار پیشبینی (y_pred) خروجی های متناظر با این index را در جدا میکنیم. در حلقه بعد میخواهیم مشاهده کنیم به ازای به ازای این label های اصلی، به صورت تک تک از هر label چه تعدادی پیشبینی شده و در label_predictions_count می گذاریم.

مثال: در هنگام اجرای اول حلقه، از میان labels = [0, 1, 2]، label=0 انتخاب میشود، سپس ایندکس جاهایی که در y_true، درایه صفر دارد جدا شده و در label_indices قرار داده میشود. سپس داخلی یک بار به ازای predicted_label=0 تعداد پیشبینی های صفر در صورتی که لیبل واقعی 0 باشد محاسبه شده و در label_predictions_count اضافه میشود. بار دیگر تعداد پیشبینی های یک و بار دیگر تعداد پیشبینی های دوم.

بنابراین با هر بار اتمام حلقه داخلی، یک سطر از Confusion Matrix (بار فرض بر این که سطر ها Actual و ستون ها Predicted باشند) ساخته خواهد شد در آخر حلقه اول این سطر به مارتیس اضافه میگردد.

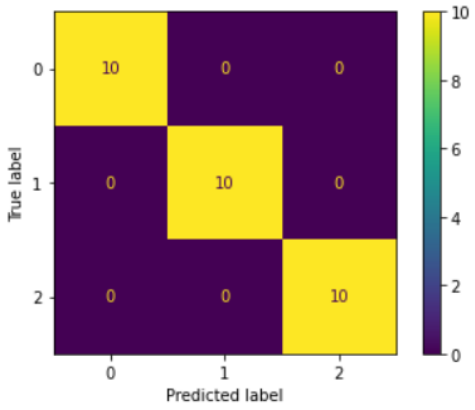
Random Forest

```
In [18]: data3 = pd.read_csv('data/data3.csv')
         data3['species'] = pd.factorize(data3['species'])[0] # Encode categories to 0, 1, and 2

In [19]: train_set, test_set = train_test_split(data3.to_numpy(), test_size=0.2, random_state=3)

In [20]: model = RandomForest(train_set, n_trees=10, seed=5)

         model.get_accuracy(test_set)

Out[20]: 100.0
```

برای اجرا، data3.csv را میخوانیم و ستون آخر را نیز به صفرو یک و ۲ انکود میکنیم. سپس داده های آموزشی و تست را به صورت 80% و 20% جدا میکنیم و مدل را آموزش میدهیم. مشاهده میکنیم که دقت ۱۰۰ درصد بدست میآید.

Confusion Matrix

In [21]:
```
y_pred = model.predict(test_set)
y_true = test_set[:, -1]

confusion_matrix = model.create_confusion_matrix(test_set[:, -1], y_pred)
disp = ConfusionMatrixDisplay(np.array(confusion_matrix))
disp.plot();
```



برای رسم Confusion Matrix نیز با کمک از ConfusionMatrixDisplay از sklearn نتیجه فوق حاصل میشود. با توجه به دقت ۱۰۰ درصد، ماتریس به شکل قطری خواهد شد.

Impact of random tree numbers

In [22]:
```
train_set, test_set = train_test_split(data3.to_numpy(), test_size=0.2, random_state=3)
```

In [23]:
```
n_trees_values = np.arange(1, 10)
accuracies = []
for n_trees in n_trees_values:
    model = RandomForest(train_set, n_trees=n_trees, seed=5)
    accuracies.append(model.get_accuracy(test_set))
```

In [24]:
```
accuracies
```

Out[24]:
```
[93.33333333333333,
 93.33333333333333,
 96.66666666666667,
 96.66666666666667,
 100.0,
 100.0,
 96.66666666666667,
 100.0,
 100.0]
```

برای تاثیر تعداد Random Tree نیز هنگام ساخت داده های آموزشی و تست از random_state=3 و هنگام آموزشRandomForest نیز از seed=5 استفاده میکنیم. چرا که در صورت عدم استفاده از این ۲ مورد حتی با ثابت نگه داشتن تعداد درخت نتیجه متفاوت خواهد شد. پس با ثابت نگه داشتن این ۲ مورد مشاهده میکنیم با افزایش تعداد درخت از ۱تا۱۰، گاهی افزایش در دقت و گاهی نیز کاهش داریم اما به طور کلی میتوان گفت با افزایش تعداد درخت تصادفی، دقت معمولا افزایش و زمان اجرا همواره طولانی تر خواهد شد.

۳- متاسفانه به دلیل کمبود وقت و وجود کوییز از سایر دروس، پیاده سازی AdaBoost صورت نگرفته است و از sklearn کمک گرفته شده است.

AdaBoost

```
In [25]: data4 = pd.read_csv('data/data4.csv')
         data4.head(10)
```

Out[25]:

	PassengerId	Pclass	Sex	Age	Target_class
0	1	3	male	22.0	0
1	2	1	female	38.0	1
2	3	3	female	26.0	1
3	4	1	female	35.0	1
4	5	3	male	35.0	0
5	6	3	male	NaN	0
6	7	1	male	54.0	0
7	8	3	male	2.0	0
8	9	3	female	27.0	1
9	10	2	female	14.0	1

```
In [26]: data4.drop("PassengerId", axis=1, inplace=True)
         data4.head(10)
```

Out[26]:

	Pclass	Sex	Age	Target_class
0	3	male	22.0	0
1	1	female	38.0	1
2	3	female	26.0	1
3	1	female	35.0	1
4	3	male	35.0	0
5	3	male	NaN	0
6	1	male	54.0	0
7	3	male	2.0	0
8	3	female	27.0	1
9	2	female	14.0	1

برای پیش پردازش مشاهده میکنیم که PassengerId کمکی در پیشبینی نمیکند. لذا حذف میکنیم.

از طرفی تعدادی درایه nan داریم و سطر های آن ها را نیز حذف میکنیم:

```
In [27]: data4.dropna(inplace=True)
         data4.head(10)
```

Out[27]:

	Pclass	Sex	Age	Target_class
0	3	male	22.0	0
1	1	female	38.0	1
2	3	female	26.0	1
3	1	female	35.0	1
4	3	male	35.0	0
6	1	male	54.0	0
7	3	male	2.0	0
8	3	female	27.0	1
9	2	female	14.0	1
10	3	female	4.0	1

به توجه به این که Age یک ویژگی عددی است، آن را به صورت زیر انکود میکنیم:

Age

- age <= 19: **Teenager**
- 20 <= age < 50: **Adult**
- 50 <= age < 65: **Middle Age**
- 65 <= age: **Elderly**

```
In [28]: age = data4['Age'].copy()
age.loc[data4['Age'] <= 19] = "Teenager"
age.loc[(20 <= data4['Age']) & (data4['Age'] < 50)] = "Adult"
age.loc[(50 <= data4['Age']) & (data4['Age'] < 65)] = "Middle Age"
age.loc[data4['Age'] >= 65] = "Elderly"
data4['Age'] = age

data4.head(10)
```

Out[28]:

	Pclass	Sex	Age	Target_class
0	3	male	Adult	0
1	1	female	Adult	1
2	3	female	Adult	1
3	1	female	Adult	1
4	3	male	Adult	0
6	1	male	Middle Age	0
7	3	male	Teenager	0
8	3	female	Adult	1
9	2	female	Teenager	1
10	3	female	Teenager	1

و در آخر به کمک One-hot encoder تمام ستون ها به جز ستون آخر را encode میکنیم:

Encoding categorical data using one-hot encoder

In [29]:
```
categorical_features = [column for column in data4 if column != 'Target_class']

data4 = pd.get_dummies(data4, columns=categorical_features)

# move Target_class to the last column
data4 = data4[[column for column in data4 if column != 'Target_class'] + ['Target_class']]

data4.head(10)
```

Out[29]:

	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	Age_Adult	Age_Elderly	Age_Middle Age	Age_Teenager	Target_class
0	0	0	1	0	1	1	0	0	0	0
1	1	0	0	1	0	1	0	0	0	1
2	0	0	1	1	0	1	0	0	0	1
3	1	0	0	1	0	1	0	0	0	1
4	0	0	1	0	1	1	0	0	0	0
6	1	0	0	0	1	0	0	1	0	0
7	0	0	1	0	1	0	0	0	1	0
8	0	0	1	1	0	1	0	0	0	1
9	0	1	0	1	0	0	0	0	1	1
10	0	0	1	1	0	0	0	0	1	1

همانطور که گفته شد، برای AdaBoost از sklearn کمک گرفته شده به طوری که صد Classifier داشته باشد. مشاهده میکنیم که دقت ۷۴ درصد برای آن حاصل شده و confusion_matrix نیز به نحو زیر میباشد:

In [30]:
```
X = data4.drop('Target_class', axis=1)
y = data4['Target_class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [31]:
```
clf = AdaBoostClassifier(n_estimators=100, random_state=0)

clf.fit(X_train, y_train)

accuracy = clf.score(X_test, y_test)
accuracy
```

Out[31]:
```
0.7412587412587412
```

In [32]:
```
y_predict = clf.predict(X_test)

disp = ConfusionMatrixDisplay(confusion_matrix(y_test, y_predict))
disp.plot();
```