نام و نام خانوادگی: حسنا اویارحسینی	شماره دانشجویی: ۹۸۲۳۳۰۸
نام درس: مقدماتی بر یادگیری ماشین	شماره تمرین: تمرین شماره ۴

توضیح کد:

ابتدا داده را لود میکنیم با توجه به اینکه سه ستون 'Family','Genus', 'Species' عددی نیستند لازم است ابتدا آنها را encode کنیم برای این کار از LabelEncoder استفاده میکنیم و داده های ستون family را اینکود میکنیم و داده های دو ستون دیگر را حذف میکنیم زیرا لیبل هایی هستند که ما در این سوال به آنها نیاز نداریم.

پس از انجام این پیش پردازش ها داده های تست و آموزش را میسازیم. حال که داده ها آماده شد به حل بخش های مختلف سوال میپردازیم.

```
cont_cols = df.columns.difference(['Family','Genus', 'Species'])
cont_cols_df = df[cont_cols]

# label encoding categorical features (str-->float)
le = LabelEncoder()
le.fit(df["Family"])
cat_cols_arr = le.transform(df["Family"])
cat_cols_df = pd.DataFrame(cat_cols_arr, columns=["Family"])

# merge cont&cat dfs
df = pd.concat([cont_cols_df, cat_cols_df], axis=1)
```

```
X = df.loc[:, df.columns != 'Family']
y = df['Family']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
X_train.shape, X_test.shape
```

```
((5036, 23), (2159, 23))
```

```
df.head()
```

	MFCCs_1	MFCCs_2	MFCCs_3	MFCCs_4	MFCCs_5	MFCCs_6	MFCCs_7	MFCCs_8	MFCCs_9	MFCCs_10	...	MFCCs_15	MFCCs_16	MFCCs_17	MFCCs_18	MFCCs_19	MFCCs_20	MF
0	1.0	0.152936	-0.105586	0.200722	0.317201	0.260764	0.100945	-0.150063	-0.171128	0.124676	...	0.135752	-0.024017	-0.108351	-0.077623	-0.009568	0.057684	0
1	1.0	0.171534	-0.098975	0.268425	0.338672	0.268353	0.060835	-0.222475	-0.207693	0.170883	...	0.163320	0.012022	-0.090974	-0.056510	-0.035303	0.020140	0
2	1.0	0.152317	-0.082973	0.287128	0.276014	0.189867	0.008714	-0.242234	-0.219153	0.232538	...	0.207338	0.083536	-0.050691	-0.023590	-0.066722	-0.025083	0
3	1.0	0.224392	0.118985	0.329432	0.372088	0.361005	0.015501	-0.194347	-0.098181	0.270375	...	0.100413	-0.050224	-0.136009	-0.177037	-0.130498	-0.054766	-0

الف) در بخش اول hard svm خطی به صورت زیر و به کمک sklearn میسازیم و سپس دقت مدل را بر روی داده های تست و آموزش به کمک تابع calculate_accuracy بدست میاوریم و تعداد support vector ها را نیز به کمک متغیر support_vectors_ چاپ میکنیم:

```
def calculate_accuracy(clf, X, actual_classes):
    predicted_classes = clf.predict(X)
    accuracy = accuracy_score(actual_classes, predicted_classes)
    return accuracy * 100
```

---

a)

```
clf = svm.SVC(kernel="linear")
clf.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

```
print(f"accuracy on train: {calculate_accuracy(clf, X_train, y_train)} %")
print(f"accuracy on test: {calculate_accuracy(clf, X_test, y_test)} %")
```

```
accuracy on train: 97.1604447974583 %
accuracy on test: 95.87772116720704 %
```

```
print("number of support vectors:")
print(len(clf.support_vectors_))
```

```
number of support vectors:
597
```

ب) در این بخش یک soft svm میخواهیم برای پیدا کردن پارامتر بهینه این مدل که C باشد نیاز به داده validation داریم پس ابتدا داده ها را به سه دسته تست و آموزش و اعتبار سنجی تقسیم میکنیم:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.3, random_state=42)
```

```
X_train.shape, X_test.shape, X_val.shape
```

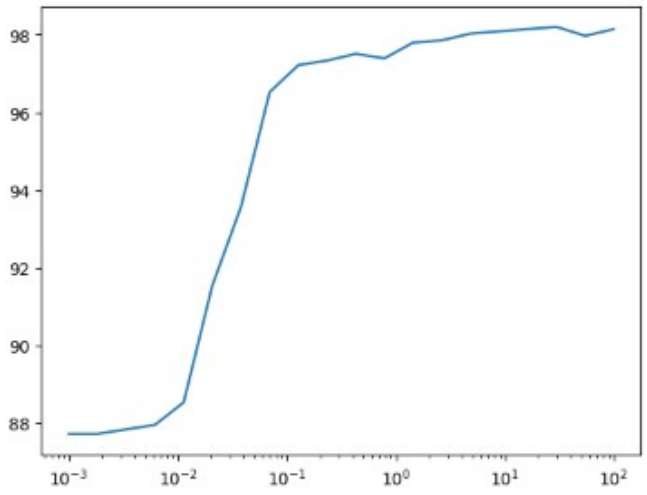
```
((4029, 23), (1439, 23), (1727, 23))
```

سپس با ازای ۲۰ مقدار مختلف برای C که به صورت لگاریتمی در بازه ۰.۰۰۱ تا ۱۰۰ تغییر میکند مدل را روی داده های آموزش آموزش میدهیم و دقت را بر روس دادگان اعتبار سنجی محاسبه میکنیم و نمودار دقت به C را رسممیکنیم و بهترین C که دقت ماکسیمم را میدهد پیدا میکنیم:

```
result = []
C_s = list(np.logspace(-3, 2, 20))
for c in C_s:
    clf = svm.SVC(kernel="linear", C=c)
    clf.fit(X_train, y_train)
    result.append(calculate_accuracy(clf, X_val, y_val))
```

```
fig, ax0 = plt.subplots()
ax0.set_xscale('log')
ax0.plot(C_s, result)
```

```
[<matplotlib.lines.Line2D at 0x7f542c22ecb0>]
```




```
C_best = C_s[np.argmax(result)]
print(f"best C = {C_best}")
```

```
best C = 29.763514416313193
```

در نهایت مدل را با بهترین C که پیدا کردیم روی داده های آموزشی آموزش میدهیم و نتایج را بر روی دادگان تست و آموزش بررسی میکنیم:

```
C_best = C_s[np.argmax(result)]
print(f"best C = {C_best}")
```

```
best C = 29.763514416313193
```

```
clf = svm.SVC(kernel="linear", C=C_best)
clf.fit(X_train, y_train)
```

```
SVC(C=29.763514416313193, kernel='linear')
```

```
print(f"accuracy on train: {calculate_accuracy(clf, X_train, y_train)} %")
print(f"accuracy on test: {calculate_accuracy(clf, X_test, y_test)} %")
```

```
accuracy on train: 97.74137503102507 %
accuracy on test: 97.28978457261988 %
```

```
print("number of support vectors:")
print(len(clf.support_vectors_))
```

```
number of support vectors:
322
```

پ) این بار hard svm غیر خطی را با دو کرنل poly , rbf آموزش میدهیم برای این کار کافیست نوع کرنل را در هنگام تعریف مدل تعیین کنیم و مانند بخش الف مدل را آموزش و سپس دقت آن را محاسبه کنیم:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
clf = svm.SVC(kernel="rbf")
clf.fit(X_train, y_train)
print(f"accuracy on train: {calculate_accuracy(clf, X_train, y_train)} %")
print(f"accuracy on test: {calculate_accuracy(clf, X_test, y_test)} %")
print("number of support vectors:")
print(len(clf.support_vectors_))
```

```
accuracy on train: 87.5297855440826 %
accuracy on test: 86.24363131079204 %
number of support vectors:
1395
```

```
clf = svm.SVC(kernel="poly")
clf.fit(X_train, y_train)
print(f"accuracy on train: {calculate_accuracy(clf, X_train, y_train)} %")
print(f"accuracy on test: {calculate_accuracy(clf, X_test, y_test)} %")
print("number of support vectors:")
print(len(clf.support_vectors_))
```

```
accuracy on train: 85.34551231135822 %
accuracy on test: 84.52987494210282 %
number of support vectors:
1815
```

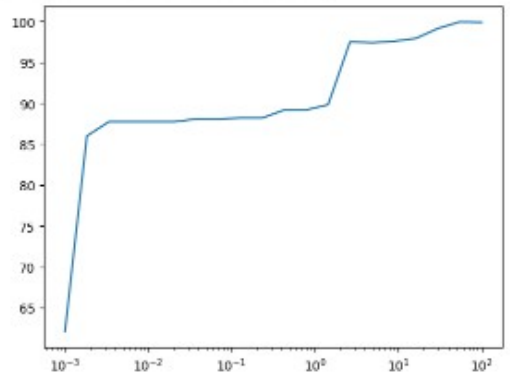
مشاهده میکنیم که RBF دقت بیشتری به ما میدهد پس برای بخش های بعد از آن استفاده میکنیم.

ت) در این بخش soft svm غیرخطی با کرنل rbf را میخواهیم و مانند بخش ب برای پیدا کردن پارامتر های بهینه از دادگان اعتبارسنجی استفاده میکنیم پس با همان روش توضیح داده شده در بخش ب بهترین C را به کمک دادگان اعتبارسنجی پیدا میکنیم و برای آن مدل نهایی را آموزش میدهیم و نتایجش را بررسی میکنیم:

```
result = []
C_s = list(np.logspace(-3, 2, 20))
for c in C_s:
    clf = svm.SVC(kernel="rbf", C=c)
    clf.fit(X_train, y_train)
    result.append(calculate_accuracy(clf, X_val, y_val))
```

```
fig, ax0 = plt.subplots()
ax0.set_xscale('log')
ax0.plot(C_s, result)
```

```
[<matplotlib.lines.Line2D at 0x7f542c0582e0>]
```




```
C_best = C_s[np.argmax(result)]
print(f"best C = {C_best}")
```

```
best C = 54.555947811685144
```

```
clf = svm.SVC(kernel="rbf", C=C_best)
clf.fit(X_train, y_train)
print(f"accuracy on train: {calculate_accuracy(clf, X_train, y_train)} %")
print(f"accuracy on test: {calculate_accuracy(clf, X_test, y_test)} %")
print("number of support vectors:")
print(len(clf.support_vectors_))
```

```
accuracy on train: 99.82625961777116 %
accuracy on test: 99.79152189020152 %
number of support vectors:
341
```

ث) در بخش آخر میخواهیم بخش ت را با ۴ fold cross validation انجام دهیم برای این کار از تابع زیر که همانند تابع تمرین قبلی است استفاده میکنیم:

```python
def cross_val_predict(model, kfold : KFold, X : np.array, y : np.array):

    model_ = cp.deepcopy(model)

    actual_classes = np.empty([0], dtype=int)
    predicted_classes = np.empty([0], dtype=int)
    accuracy = np.empty([0], dtype=int)

    for train_ndx, test_ndx in kfold.split(X):
        # Extracts the rows from the data for the training and testing
        train_X, train_y, test_X, test_y = X[train_ndx], y[train_ndx], X[test_ndx], y[test_ndx]
        # Appends the actual target classifications to actual_classes
        actual_classes = np.append(actual_classes, test_y)
        # Fits the machine learning model using the training data extracted from the current fold
        model_.fit(train_X, train_y)
        # Uses the fitted model to predict the target classifications for the test data in the current fold
        predicted_classes = np.append(predicted_classes, model_.predict(test_X))
        accuracy = np.append(accuracy, accuracy_score(actual_classes, predicted_classes))

    return actual_classes, predicted_classes, accuracy.mean(), model_
```

این تابع در یک حلقه هر بار یک فولد را به عنوان داده تست میگیرد و با بقیه فولد ها مدل را آموزش میدهد و نتایج حاصل از پیش بینی بر روی داده های تست را در آرایه predicted_class ذخیره میکند. در آخر نیز دقت را با توجه به برچسب های اولیه برای این داده محاسبه و ذخیره میکند. در نهایت این تابع کلاس های صحیح و پیش بینی شده و دقت میانگین را برمیگرداند.
بعد از پیدا کردن بهترین C با روش توضیح داده شده در بخش ب مدل را این بار با تابعی که تعریف کردیم و به کمک cross validatio روی تمام داده ها آموزش میدهیم و نتیجه را بررسی میکنیم:

```
C_best = C_s[np.argmax(result)]
print(f"best C = {C_best}")
```

```
best C = 54.555947811685144
```

```
model = svm.SVC(kernel="rbf", C=C_best)
kfold = KFold(n_splits=5, random_state=42, shuffle=True)
actual_classes, predicted_classes, accuracy, clf = cross_val_predict(model, kfold, X.to_numpy(), y.to_numpy())

print(f"accuracy on train: {calculate_accuracy(clf, X_train, y_train)} %")
print(f"accuracy on test: {calculate_accuracy(clf, X_test, y_test)} %")
print("number of support vectors:")
print(len(clf.support_vectors_))
```

```
accuracy on train: 99.8758997269794 %
accuracy on test: 99.93050729673384 %
number of support vectors:
398
```

مقایسه تمامی نتایج:

	Linear hard svm	Linear soft svm	Hard svm with rbf kernel	Hard svm with poly kernel	Soft svm with rbf kernel	Soft svm with rbf kernel and cross validation
Accuracy on train	۹۷.۱۶۰۴۴۴۷ ۹۷۴۵۸۳	۹۷.۷۴۱۳۷۵۰ ۳۱۰۲۵۰۷	۸۷.۵۲۹۷۸۵۵ ۴۴۰۸۲۶	۸۴.۵۲۹۸۷۴۹ ۴۲۱۰۲۸۲	۹۹.۸۲۶۲۵۹۶ ۱۷۷۷۱۱۶	۹۹.۸۷۵۸۹۹۷ ۲۶۹۷۹۴
Accuracy on test	۹۵.۸۷۷۷۲۱۱ ۶۷۲۰۷۰۴	۹۷.۲۸۹۷۸۴۵ ۷۲۶۱۹۸۸	۸۶.۲۴۳۶۳۱۳ ۱۰۷۹۲۰۴	۸۵.۳۴۵۵۱۲۳ ۱۱۳۵۸۲۲	۹۹.۷۹۱۵۲۱۸ ۹۰۲۰۱۵۲	۹۹.۹۳۰۵۰۷۲ ۹۶۷۳۳۸۴
Numberofsupport vectos	۵۹۷	۳۲۲	۱۳۹۵	۱۸۱۵	۳۴۱	۳۹۸

مقایسه دقت روی داده آموزش:

Soft svm with rbf kernel > Soft svm with rbf kernel and cross validation > Linear soft svm > Linear hard svm > Hard svm with rbf kernel > Hard svm with poly kernel

مقایسه دقت روی داده تست:

Soft svm with rbf kernel and cross validation > Soft svm with rbf kernel > Linear soft svm > Linear hard svm > Hard svm with rbf kernel > Hard svm with poly kernel

مشاهده میکنیم که دقت مدل های سافت به علت انعطاف بیشتر بهتر است و همچنین استفاده از کرنل rbf و cross

validation عملکرد را بهبود میبخشد در روش های هارد اما مدل خطی عملکرد بهتری داشته است.
استفاده از cross validation باعث شد دقت مدل بر روی داده های تست افزایش پیدا کند اما کمی روی داده های آموزشی دقت کمتری دارد.

مقایسه تعداد support vectors:

Hard svm with poly kernel > Hard svm with rbf kernel > Linear hard svm > Soft svm with rbf kernel and cross validation > Soft svm with rbf kernel > Linear soft svm

مشاهده میکنیم که تعداد support vector ها در روش های هارد بسیار بیشتر از زمانی است که مدل را سافت میکنیم زیرا در حالت سافت مدل انعطاف پذیر تری است.