hw07

November 24, 2023

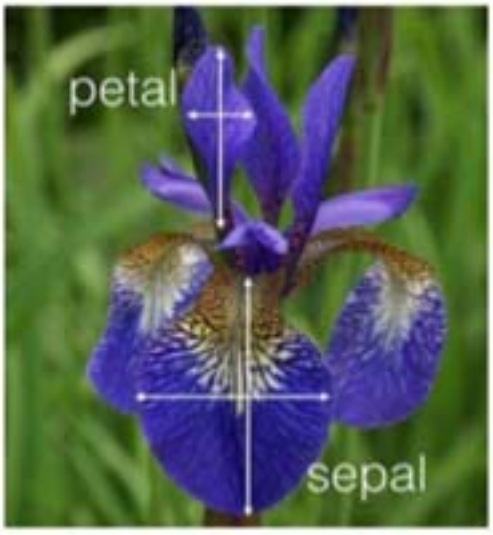
1 Homework 7 : Linear / Nonlinear Classification

INDIVIDUAL HOMEWORK The homework is not collaborative anymore. Please respect the academic integrity. **Remember: if you get caught on cheating, you get F.**

Each subproblem is worth 10 pts. All cells are marked with instructions to insert your code. Please complete all cells as directed.

What to turn in: - Please print the notebook containing the answers and results into a pdf file (you can use `File - Print`). Submit this pdf file to the main homework entry in gradescope. Be sure to locate your answers for each problem when you submit, as ususal. In the worst case where you cannot print it into a pdf file somehow, you can create a Microsoft word document and then copy-paste screenshots showing your code and output parts by parts. - You also need to submit this jupyter notebook file filled with your answers in the code entry in gradescope.

Description: This homework will study 3-class classification in the famous "Iris" dataset. The dataset was introduced by the British statistician and biologist Ronald Fisher in his 1936 paper "The use of multiple measurements in taxonomic problems" as an example of linear discriminant analysis. The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear classifier to distinguish the species from each other. We will do the same using classifiers that we have learned.



[2]:
```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.pipeline import Pipeline
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
import warnings

# Suppress warnings
warnings.filterwarnings("ignore")
np.random.seed(0)
```

1.1 Problem 1 : Load / Explore Data

1.1.1 (a)

Because this dataset is so well-known, Scikit-Learn includes a special function for loading it, which is provided below. Do the following in the cell below: * Load the data and create a Train / Test split with 25% test data (train_test_split() function) * For the above, make sure to use the provided random state so that results are repeatable * Display the training inputs (you can use function display())

Note: You will need the feature names later on. It is helpful at this point to store them in a set using the DataFrame.colums property.

[18]:
```
# use this random state for train/test split
random_state=1234

iris = datasets.load_iris(as_frame=True)
X = iris.data
y = iris.target

# Insert code here
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=random_state)
features = []
for feature in X_train.columns:
    features.append(feature)
display(X_train, y_train)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
22	4.6	3.6	1.0	0.2
93	5.0	2.3	3.3	1.0
36	5.5	3.5	1.3	0.2
68	6.2	2.2	4.5	1.5
32	5.2	4.1	1.5	0.1

```
..                 …                …                …                …
143                6.8                3.2                5.9                2.3
116                6.5                3.0                5.5                1.8
53                 5.5                2.3                4.0                1.3
38                 4.4                3.0                1.3                0.2
47                 4.6                3.2                1.4                0.2

[112 rows x 4 columns]
```

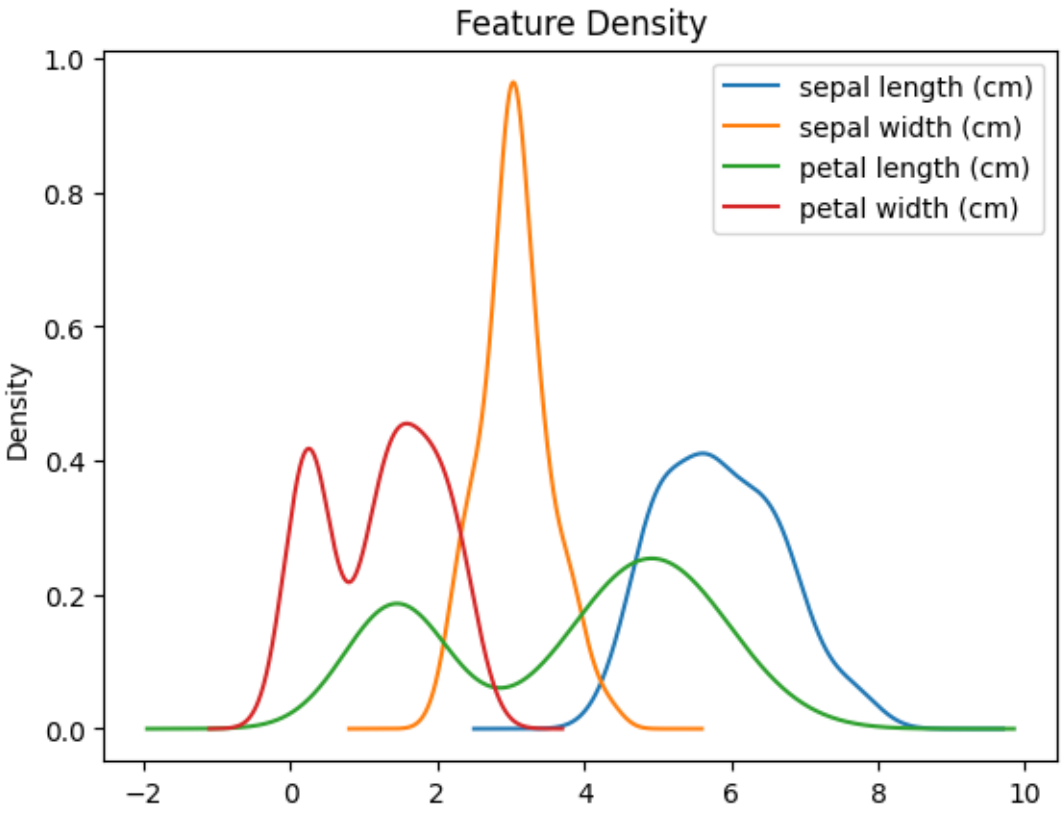
```
22     0
93     1
36     0
68     1
32     0
      ..
143    2
116    2
53     1
38     0
47     0
Name: target, Length: 112, dtype: int64
```

1.1.2 (b)

Now we will explore our feature distributions. In the cell below, use the Pandas DataFrame plot feature to plot the density of each feature in the training data.

Documentation - Pandas - DataFrame.plot

[20]:
```
# Insert code here
plot = X_train.plot(kind='density', title='Feature Density')
```

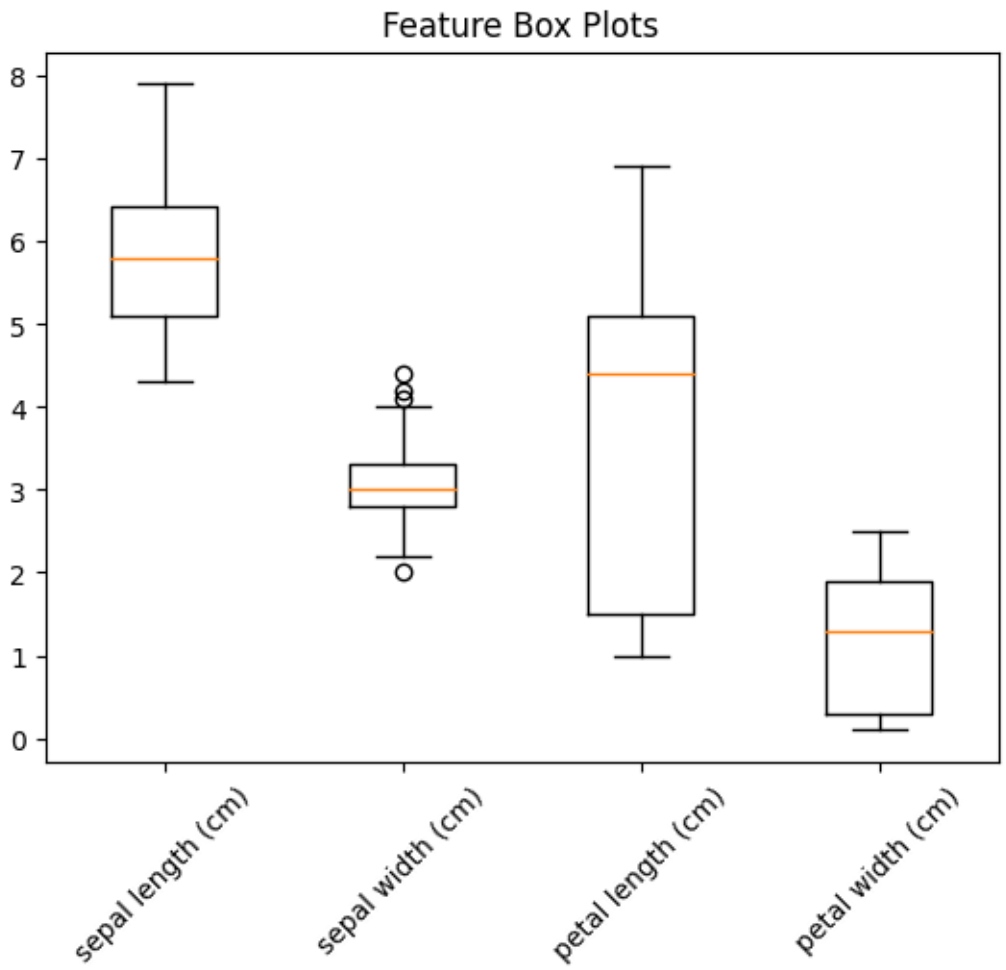


1.1.3 (c)

Sometimes it is better to look at distributions of each feature plotted together. In the cell below produce a boxplot (use plt.boxplot()) of each feature in the training data. **Make sure to rotate X-tick labels 45-degrees so they are readable.**

[25]:
```
# Insert code here
plt.boxplot(X_train, labels=features)
plt.title('Feature Box Plots')
plt.xticks(rotation=45)
```

[25]:
```
(array([1, 2, 3, 4]),
 [Text(1, 0, 'sepal length (cm)'),
  Text(2, 0, 'sepal width (cm)'),
  Text(3, 0, 'petal length (cm)'),
  Text(4, 0, 'petal width (cm)')])
```



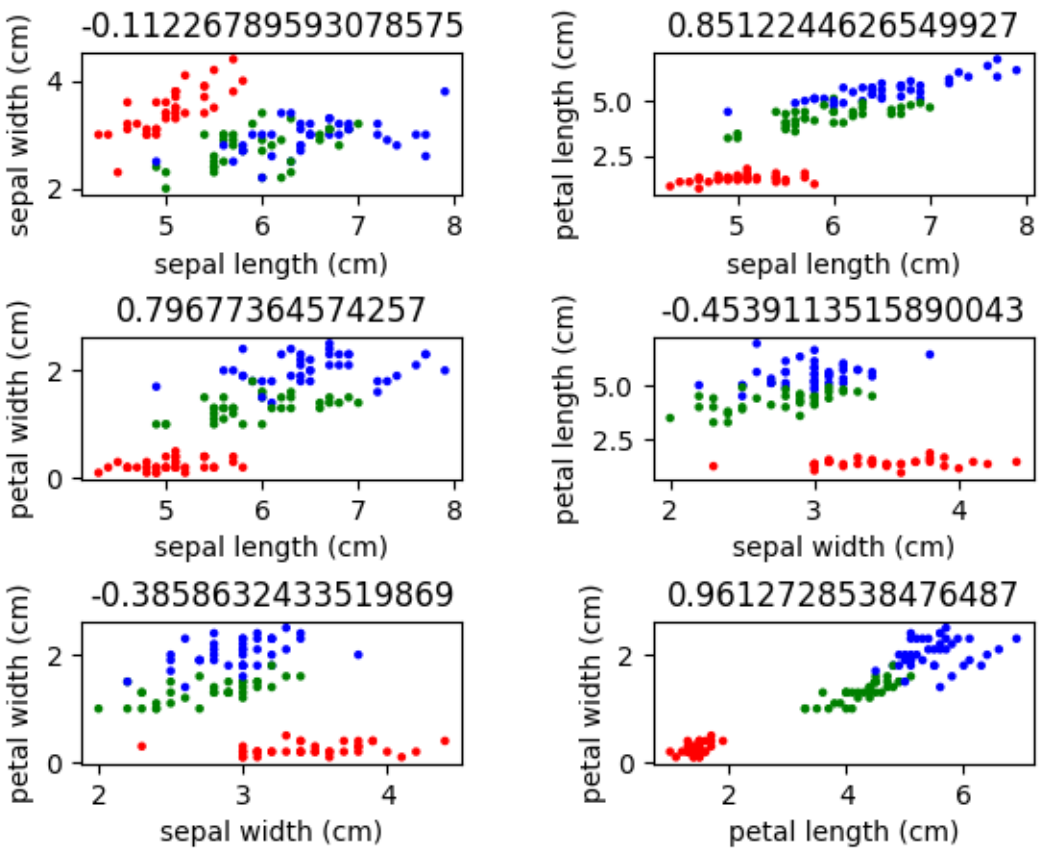
1.1.4 (d)

Now let's see how well we can separate classes from each pair of features. In the cell below produce a scatterplot of **every pair of features** in the training data. There will be 6 scatterplots in all. Make sure to follow these instructions: * Color each marker red, green, or blue depending on the true class label * Use **numpy.corrcoef** to compute the correlation coefficient of each feature * Title each plot with the correlation coefficient * Label each axis using the corresponding feature name

[109]:
```
# Insert code here
print(features)
# print(y_train.to_numpy())
fix, axs = plt.subplots(3, 2)
plt.subplots_adjust(wspace=0.5, hspace=1)

colors = ['r', 'g', 'b']
pairs = []
```

```
for i in range(len(features)):
    for j in range(i+1, len(features)):
        pairs.append((features[i], features[j]))
pair_ctr = 0
# df = df.replace([np.inf, -np.inf], np.nan).dropna()
for i in range(3):
    for j in range(2):
        pair = pairs[pair_ctr]
        x = X_train[[pair[0]]]
        y = X_train[[pair[1]]]
        # print(X_train.corr()[pair[0]][pair[1]])
        axs[i,j].scatter(x, y, s=5, c=[colors[color] for color in y_train.
 ↪to_numpy()])
        corr_coef = X_train.corr()[pair[0]][pair[1]]
        axs[i,j].set_title(corr_coef)
        axs[i,j].set_xlabel(pair[0])
        axs[i,j].set_ylabel(pair[1])
        pair_ctr += 1

plt.show()
# plt.tight_layout()
# plt.subplots_adjust(wspace=0.5, hspace=0.5)
# axs[0,0].scatter(X_train[['sepal length (cm)']], X_train[['sepal width␣
 ↪(cm)']], s=5)
# axs[0,1].scatter(X_train[['sepal length (cm)']], X_train[['sepal width␣
 ↪(cm)']])
# axs[1,1].scatter(X_train[['sepal length (cm)']], X_train[['sepal width␣
 ↪(cm)']])
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
```



1.2 Problem 2 : Train a logistic regression classifier

1.2.1 (a)

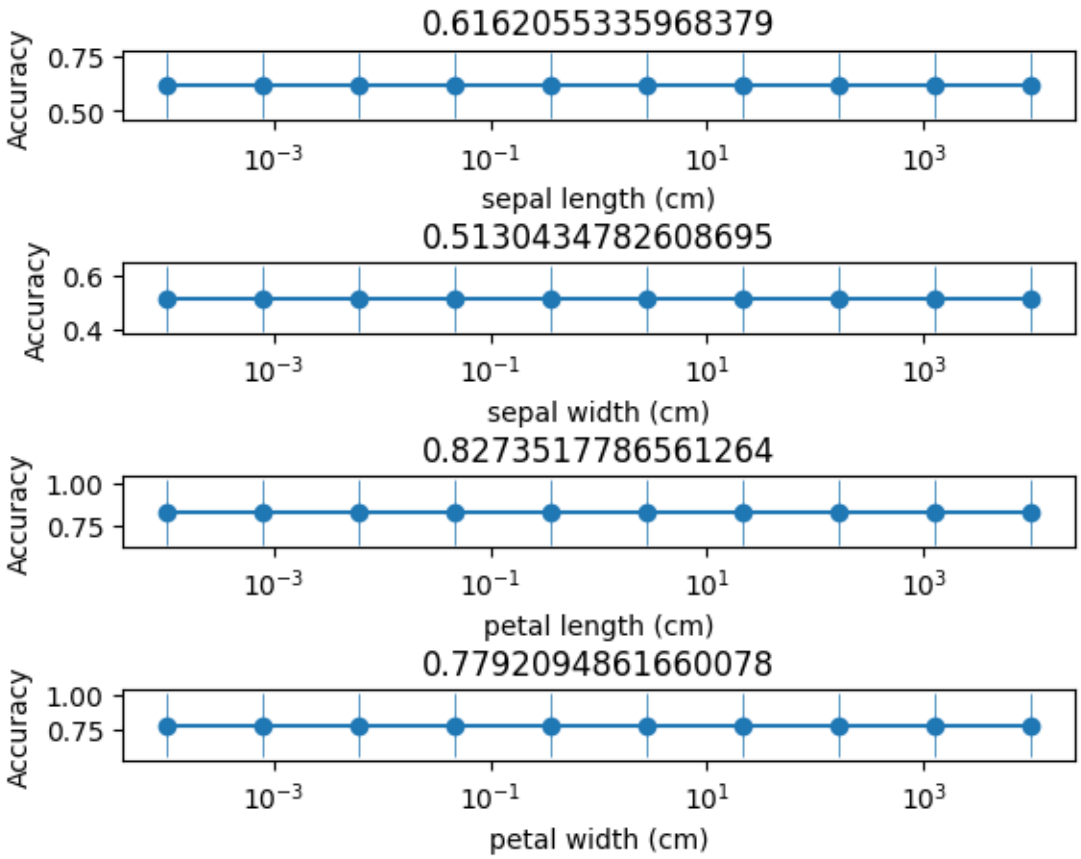
Now we will look at finding the best feature out of all the features. To do this, you will preform Cross Validation of Logistic Regression. We will break this into subproblems to walk through it. In the cell do the following: * Using LogisticRegressionCV perform 5-fold cross validation to train on each feature * For each run use Matplotlib errorbar() to plot the average +/- standard deviation of error versus regularization coefficient (the property LogisticRegressionCV.Cs_) – there should be 4 plots in total * Set plot X-label with the feature name (make the x-axis in the logarithmic scale using ax.set_xscale('log')), and Y-label "Accuracy" * Title each plot with the maximum achieved accuracy score * Report the best accuracy from cross-validation * Finally, report the best performing feature and save it for later

Make sure to set the following properties in LogisticRegressionCV: * cv=5 * max_iter=1e4 * random_state=0 * multi_class='multinomial' * Cs=10

Documentation - Scikit-Learn - LogisticRegressionCV

[205]:
```
# Insert code here
row_ctr = 0
```

```
fix, axs = plt.subplots(4,1)
plt.subplots_adjust(wspace=1, hspace=2)
best = 0
for feature in features:
    lrCV = LogisticRegressionCV(cv=5, max_iter=10000, random_state=0,␣
 ↪multi_class='multinomial', Cs=10).fit(X_train[[feature]], y_train)
    # print((lrCV.scores_))
    # print(lrCV.Cs_)
    scores = [[]] * 10
    for i in range(10):
        for l in lrCV.scores_[0]:
            scores[i].append(l[i])
    std = [0] * 10
    avg = [0] * 10
    for i in range(len(scores)):
        std[i] = np.std(scores[i])
        avg[i] = np.mean(scores[i])
    max_avg = np.mean(avg)
    plt.xscale('log')
    axs[row_ctr].errorbar(lrCV.Cs_, avg, std, fmt='o-', elinewidth=0.5)
    axs[row_ctr].set_xscale('log')
    axs[row_ctr].set_xlabel(feature)
    axs[row_ctr].set_ylabel('Accuracy')
    axs[row_ctr].set_title(max_avg)

    row_ctr += 1
    if (max_avg > best):
        best = max_avg

plt.show()
print('Best accuracy:', best)
print('Best performing feature: petal length (cm)')
```




```
Best accuracy: 0.8273517786561264
Best performing feature: petal length (cm)
```

1.2.2 (b)

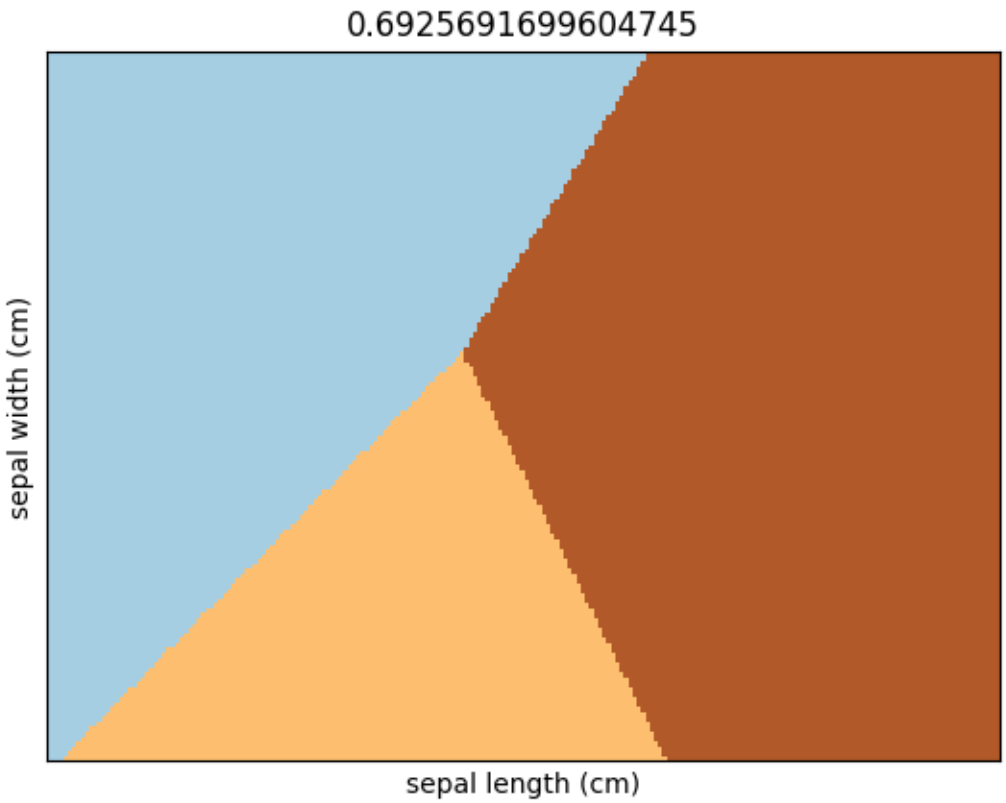
Now lets look at all pairs of features. The cell below provides a function plotLogreg2feat() to visualize the learned classifier for a pair of features. This function will draw the decision boundaries for each of the three classes, which will give us a better picture of what's going on. In the cell below that do the following: * Loop over every pair of features (there are 6 pairs total) * Using LogisticRegressionCV perform 5-fold cross validation to train a classifier on the pair of features * Make sure to use **the same cross validation options as the previous experiment** * Using plotLogreg2feat plot the learned classifier * Title each plot with the maximum average accuracy from cross validation

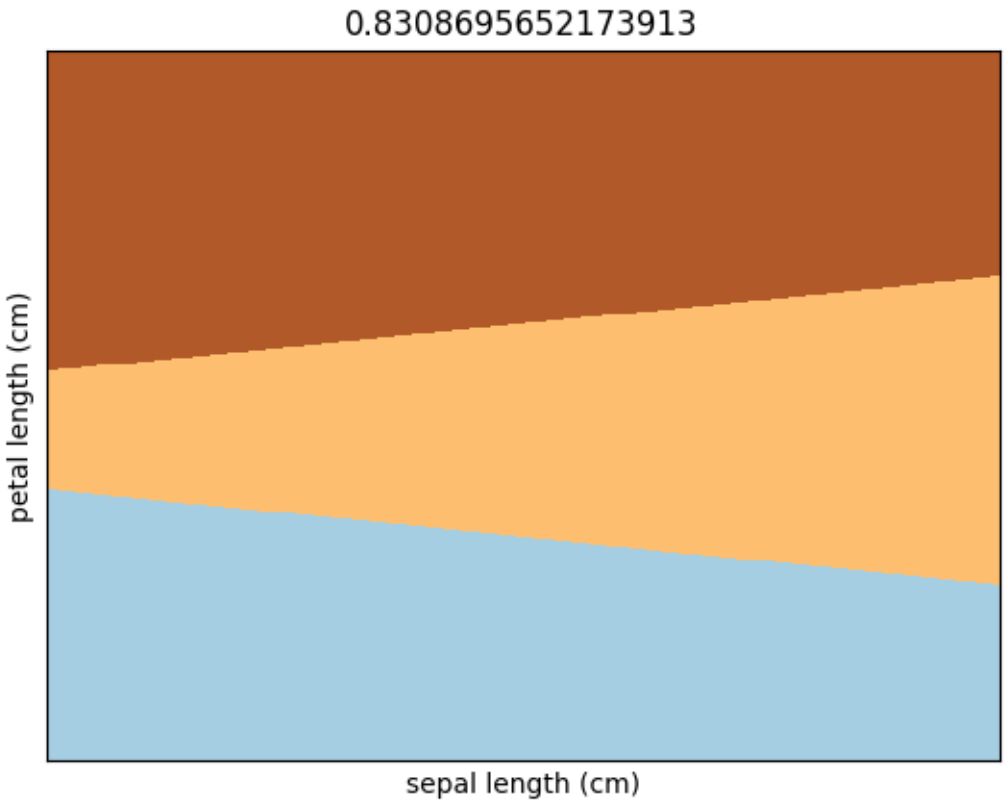
[198]:
```
def plotLogreg2feat(X, featname_1, featname_2, model):
    '''
    INPUTS:
      X - Input DataFrame (assumes Nx2 for N data points and 2 features)
      featname_1, featname_2 - String containing feature names
      model - Fitted LogisticRegressionCV model
```

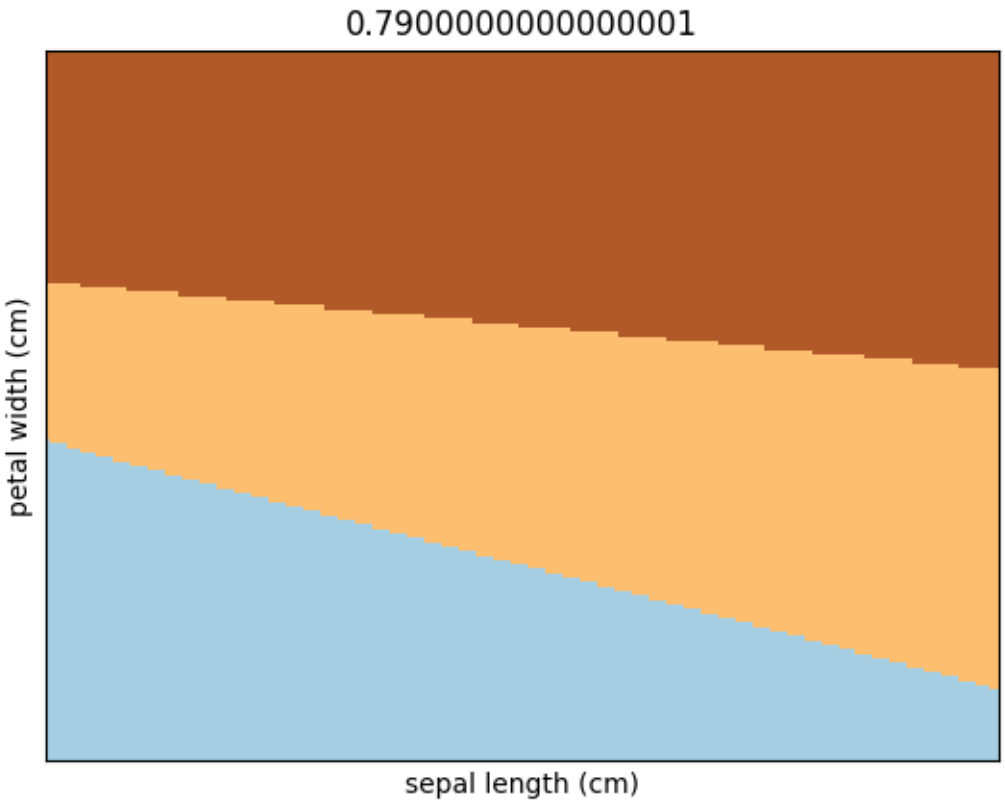
```
    OUTPUTS:
      ax - Returns figure axis object
    '''

    # make grid
    x_min, x_max = X[featname_1].min() - 0.5*X[featname_1].std(), X[featname_1].
↪max() + 0.5*X[featname_1].std()
    y_min, y_max = X[featname_2].min() - 0.5*X[featname_2].std(), X[featname_2].
↪max() + 0.5*X[featname_2].std()
    h = 0.02  # step size in the mesh
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    #plt.figure(1, figsize=(4, 3))
    fig, ax = plt.subplots()
    ax.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

    # Plot also the training points
    #plt.scatter(X_train[features[i]][ Y_train == 0 ], X_train[features[j]][␣
↪Y_train == 0 ], c='r')
    #plt.scatter(X_train[features[i]][ Y_train == 1 ], X_train[features[j]][␣
↪Y_train == 1 ], c='g')
    #plt.scatter(X_train[features[i]][ Y_train == 2 ], X_train[features[j]][␣
↪Y_train == 2 ], c='b')

    ax.scatter(X[[featname_1]][ y == 0 ], X[[featname_2]][ y == 0 ], c='r')
    ax.scatter(X[[featname_1]][ y == 1 ], X[[featname_2]][ y == 1 ], c='g')
    ax.scatter(X[[featname_1]][ y == 2 ], X[[featname_2]][ y == 2 ], c='b')

    ax.set_xlabel(featname_1)
    ax.set_ylabel(featname_2)


    #plt.xlim(xx.min(), xx.max())
    #plt.ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    return ax
```

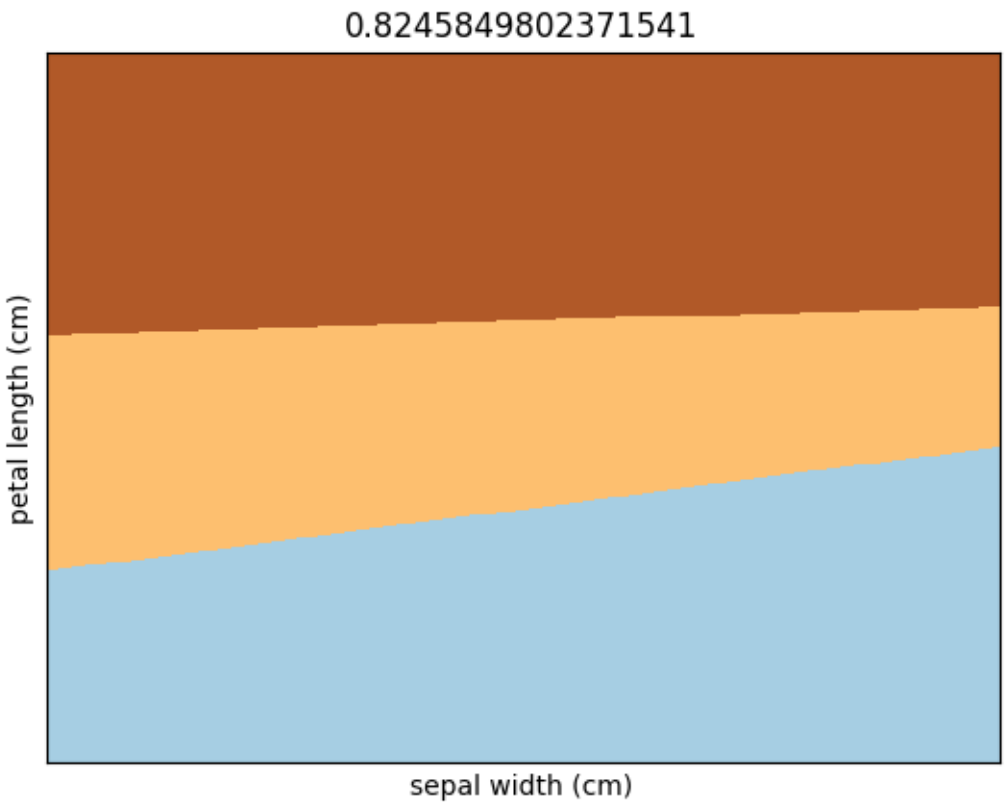
[203]:
```
# Insert code here
row = 0
for pair in pairs:
```

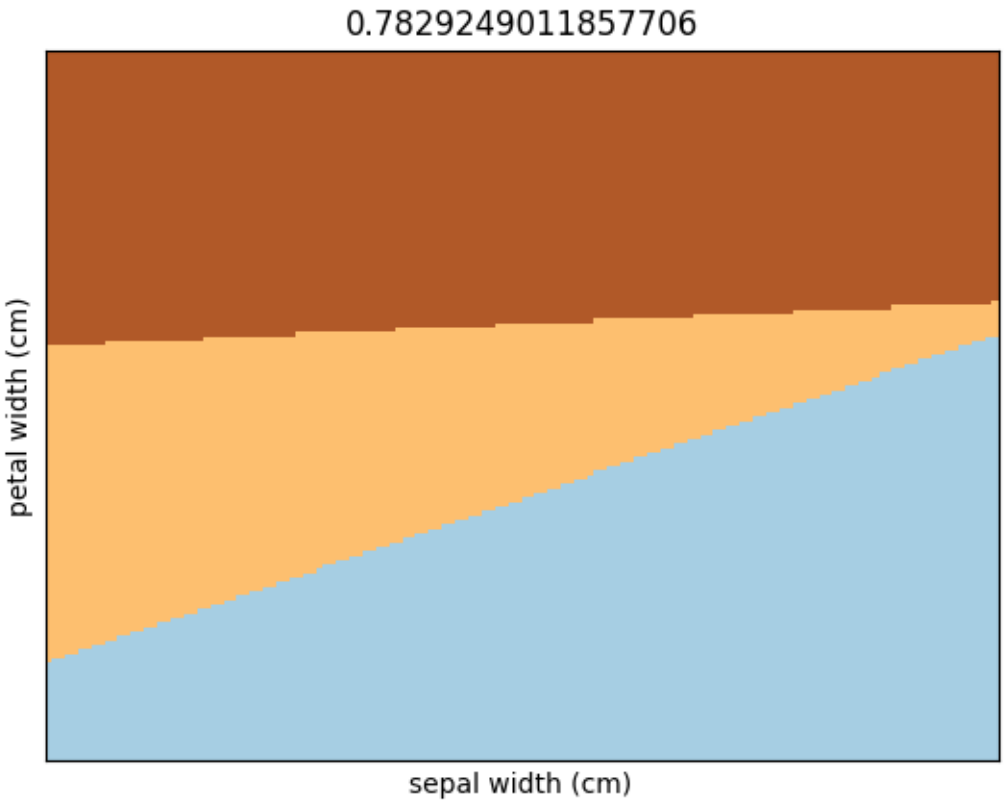
```
  lrCV = LogisticRegressionCV(cv=5, max_iter=10000, random_state=0,␣
↪multi_class='multinomial', Cs=10).fit(X_train.loc[:, [pair[0],pair[1]]],␣
↪y_train)
  scores = [[]] * 10
  for i in range(10):
      for l in lrCV.scores_[0]:
          scores[i].append(l[i])
  std = [0] * 10
  avg = [0] * 10
  for i in range(len(scores)):
      std[i] = np.std(scores[i])
      avg[i] = np.mean(scores[i])
  mean = np.mean(avg)
  plotLogreg2feat(X_train, pair[0], pair[1], lrCV).set_title(mean)
```

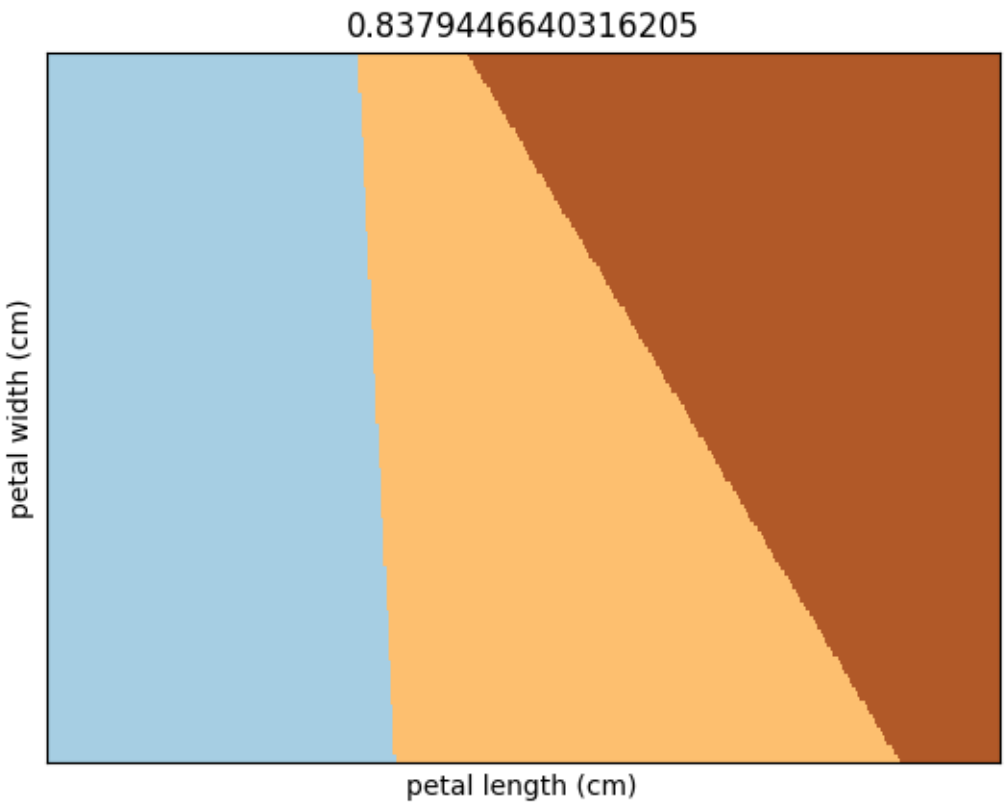



0.8308695652173913
petal length (cm)
sepal length (cm)


0.7900000000000001
petal width (cm)
sepal length (cm)


0.8245849802371541
petal length (cm)
sepal width (cm)


0.7829249011857706
petal width (cm)
sepal width (cm)



1.2.3 (c)

Surprisingly, adding pairs of features doesn't seem to improve things. Let's try training on all features. In the cell below: * Perform 5-fold cross validation (using all the same parameters as before) to train a logistic regression classifier on all features * Report the maximum of the average scores.

[213]:
```
# Insert code here
lrCV = LogisticRegressionCV(cv=5, max_iter=10000, random_state=0,␣
 ↪multi_class='multinomial', Cs=10).fit(X_train, y_train)
scores = [[]] * 10
for i in range(10):
    for l in lrCV.scores_[0]:
        scores[i].append(l[i])
std = [0] * 10
avg = [0] * 10
for i in range(len(scores)):
    std[i] = np.std(scores[i])
    avg[i] = np.mean(scores[i])
mean = np.mean(avg)
print('Max of average scores:', np.max(avg))
```

```
Max of average scores: 0.8361264822134387
```

If your results are the same as mine, the maximum score over all features is the same as over the best single feature.

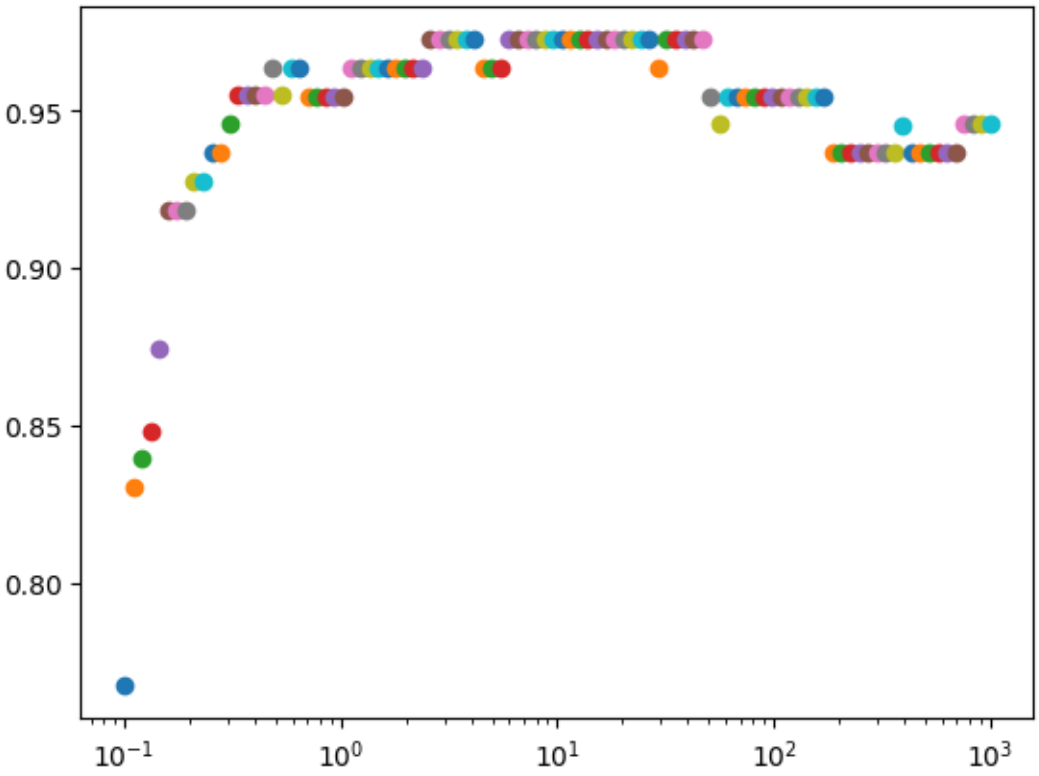
1.3 Problem 3 : Support Vector Machine

We have trained several logistic regression classifiers, all of which achieve an a cross-validation accuracy well into the 90% range. In an effort to see if we can do better, let's train one lass classifier–a support vector machine. For this classifier we will introduce a nonlinear tranformation using the Radial Basis kernel function. In the cell be low do the following: * Using Numpy.logspace create a logarithmically spaced set of 100 regularization coefficient in the range (1e-4, 1e4) * For each coefficient define a support vector classifier with kernel='rbf' and set the regularization coefficient (C=coefficient) * Perform 5-fold cross validation (e.g. using cross_val_score) * Plot the average accuracy versus regularization coefficient and report the maximum accuracy and best coefficient * Make sure to set the plot X-scale to 'log' and label axes and title

Documentation - Scikit-Learn - svm.SVC

[225]:
```
# Insert code here
rc_vals= np.logspace(-1, 3, num=100)
maxs = []
for i in range(len(rc_vals)):
    svc = SVC(C=rc_vals[i], kernel='rbf')
    score = cross_val_score(svc, X_train, y_train, cv=5)
    avg = [np.mean(avgs) for avgs in score]
    ravg = np.mean(avg)
    maxs.append(ravg)
    plt.scatter(rc_vals[i], ravg)
    plt.xscale('log')

print(np.max(maxs))
```

```
0.9727272727272727
```



My results show slightly higher accuracy under the SVM classifier. However, cross_val_score does not use the same cross validation splits as the built-in cross validation of LogisticRegressionCV (which randomizes splits). So we shall see…

1.4 Problem 4 : Evaluate on test

Now we will evaluate all classifiers on the test data. Take the best regression classifier and the best SVM classifier (with previously chosen parameters), train them, and evaluate accordingly. For each classifier report: * Test accuracy * Confusion matrix * Results of classification_report

We have only left a single cell below. Feel free to insert additional cells and arrange output as you see fit. Make sure it is readable. Feel free to explore any additional visualizations or metrics.

[ ]:
```
# Insert code here
```

[ ]:
```

```

[ ]:
```

```