Classification - Assignment 1

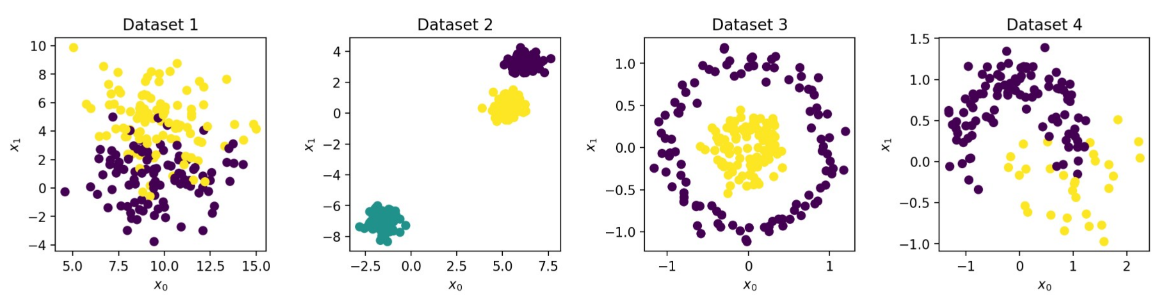
Data and Package Import

In [17]:

```
%matplotlib inline
import numpy as np
import pandas as pd
```

In [18]:

```
from sklearn.datasets import make_blobs, make_moons, make_circles
np.random.seed(4)

noisiness = 1

X_blob, y_blob = make_blobs(n_samples = 200, centers = 2, cluster_std

X_mc, y_mc = make_blobs(n_samples = 200, centers = 3, cluster_std = 0.

X_circles, y_circles = make_circles(n_samples = 200, factor = 0.3, noi

X_moons, y_moons = make_moons(n_samples = 200, noise = 0.25 * noisines

N_include = 30
idxs = []
Ni = 0
for i, yi in enumerate(y_moons):
    if yi == 1 and Ni < N_include:
        idxs.append(i)
        Ni += 1
    elif yi == 0:
        idxs.append(i)

y_moons = y_moons[idxs]
X_moons = X_moons[idxs]

fig, axes = plt.subplots(1, 4, figsize = (15, 3), dpi = 200)

all_datasets = [[X_blob, y_blob], [X_mc, y_mc], [X_circles, y_circles]

labels = ['Dataset 1', 'Dataset 2', 'Dataset 3', 'Dataset 4']
for i, Xy_i in enumerate(all_datasets):
    Xi, yi = Xy_i
    axes[i].scatter(Xi[:, 0], Xi[:, 1], c = yi)
    axes[i].set_title(labels[i])
    axes[i].set_xlabel('$x_0$')
    axes[i].set_ylabel('$x_1$')

fig.subplots_adjust(wspace = 0.4);
import numpy as np
clrs = np.array(['#003057', '#EAAA00', '#4B8B9B', '#B3A369', '#377117'
```



1. Discrimination Lines

Derive the equation for the line that discriminates between the two classes.

Consider a model of the form:

$\bar{\bar{X}}\vec{w} > 0$ if $y_i = 1$ (class 1)

$\bar{\bar{X}}\vec{w} < 0$ if $y_i = -1$ (class 2)

where $\bar{\bar{X}} = [\overrightarrow{x_0}, \overrightarrow{x_1}, \vec{1}]$ and $\vec{w} = [w_0, w_1, w_2]$.

The equation should be in the form of $x_1 = f(x_0)$. Show your work, and/or explain the process

x0w0 + x1w1 + w2 = 0 x1 = (-w0/w1)*x0 - (w2/w1)

#Therefore, x1 deprends of x0, as seen above.

Derive the discrimination line for a related non-linear model

In this case, consider a model defined by:

$y_i = w_0 x_0 + w_1 x_1 + w_2(x_0^2 + x_1^2)$

where the model predicts class 1 if $y_i > 0$ and predicts class 2 if $y_i \leq 0$.

The equation should be in the form of $x_1 = f(x_0)$. Show your work, and/or explain the process you used to arrive at the answer.

w1x1 = yi - w0x0 - w2x0^2 + w2x1^2 w1x1 - w2x1^2 = yi - w0x0 - w2x0^2 w2x1^2-w1x1 + yi - w0x0 - w2x0^2 = 0

#quadratic formula shows that it is a function of x0 x1 = (-w1 +/- sqrt((w1^2 - 4(-w2)(yi - w0x0 - w2x0^2)))/(2w2)

Briefly describe the nature of this boundary.

What is the shape of the boundary? Is it linear or non-linear?

The decision boundary depends on how the classes may be separated. If it is able to be linearly separated with a straight line, then the boundary is linear, if the classes can still be separated with a continuous line that is not straight, then the boundary is non-linear but the classes would still be separable.

2. Assessing Loss Functions

In [19]:
```
def add_intercept(X):
    intercept = np.ones((X.shape[0], 1))
    X_intercept = np.append(intercept, X, 1)
    return X_intercept
```

In [20]:
```
def linear_classifier(X, w):
    X_intercept = add_intercept(X)
    p = np.dot(X_intercept, w)
```

Write a function that computes the loss function for the perceptron model.

The function should take the followings as arguments:

- weight vector w
- the feature matrix $\bar{\bar{X}}$
- the output vector $\vec{y}$

You may want to use functions above.

In [21]:
```
def perceptron(w, X, y): #"max cost function"
    X_intercept = add_intercept(X)
    Xb = np.dot(X_intercept,w)
    loss = sum(np.maximum(0, -y*Xb))
    return loss
```

Write a function that computes the loss function for the logistic regression model.

The function should take the followings as arguments:

- weight vector w
- the feature matrix $\bar{\bar{X}}$
- the output vector $\vec{y}$

You may want to use functions above.

In [22]:
```
#need to fix problems within max cost function
#Problems: trivial soln at w = 0, max func not differentiable
def log_reg(w, X, y): #this is the softmax function from the notes
    X_intercept = add_intercept(X)
    Xb = np.dot(X_intercept, w)
    exp_yXb = np.exp(-y * Xb)
    loss = sum(np.log(1 + exp_yXb))
    return loss
```

Minimize the both loss functions using the Dataset 3 above.

In [23]:
```
from scipy.optimize import minimize
noisiness = 1
X,y = make_circles(n_samples = 200, factor = 0.3, noise = 0.1*noisines
w = np.array([1,2,3])

percep_min = minimize(perceptron, w, args = (X,y))
w_perceptron = percep_min.x

loss_reg_min = minimize(log_reg, w, args = (X,y))
```

What is the value of the loss function for the perceptron model after optimization?

In [24]:
```
print(w_perceptron)
print('The optimized perceptron loss func is',perceptron(w_perceptron,
```

```
[1.95101589 1.79683778 2.72724758]
The optimized perceptron loss func is 0.0
```

What is the value of the loss function for the logistic regression model after optimization?

In [25]:
```
print(w_log_reg)
print('The optimized log reg is', log_reg(w_log_reg,X,y))
```

```
[16.24231948 -1.11773931 -1.6600907 ]
The optimized log reg is 69.31472797610287
```

What are the two main challenges of the perceptron loss function?

The perceptron loss function has two issues, one being that it has a trivial solution at w(vec) = 0 becomes a zero vector, and the other being that the function is not differentiable.

3. Support Vector Machine

Write a function that computes the loss function of the support vector machine model.

This functions should take the followings as arguments:

- weight vector w
- the feature matrix $\bar{\bar{X}}$
- the output vector $\vec{y}$
- regularization strength α

You may want to use `add_intercept` and `linear_classifier` functions from the Problem 2.

In [26]:
```
def svm(w, X, y, alpha):
    X_intercept = add_intercept(X)
    Xb = np.dot(X_intercept,w)
    cost = sum(np.maximum(0,1-y*Xb))
```

```
    cost += alpha*np.linalg.norm(w[1:],2)
    loss = cost
```

Evaluate the effect of regularization strength.

Optimize the SVM model for **Dataset 1**.

Search over α = [0, 1, 2, 10, 100] and assess the loss function of the SVM model.

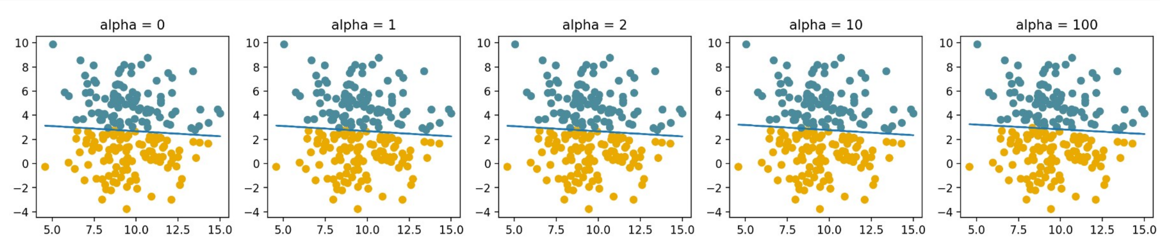
In [27]:

```
alphas = np.array([0,1,2,10,100])
w_guess = np.array([1,2,3])

for i in np.array([0,1,2,3,4]):
    cur_result = minimize(svm, w_guess, args = (X, y, alphas[i]))
    w_svm = cur_result.x
    m = -w_svm[1] / w_svm[2]
    b = -w_svm[0] / w_svm[2]
```

Plot the discrimination lines for α = [0, 1, 2, 10, 100].

In [28]:

```
alphas = np.array([0,1,2,10,100])
w_guess = np.array([1,2,3])
X = X_blob
y = y_blob * 2 - 1
fig, axes = plt.subplots(1, 5, figsize = (18, 3), dpi = 200)

for i in np.array([0,1,2,3,4]):
    cur_result = minimize(svm, w_guess, args = (X, y, alphas[i]))
    w_svm = cur_result.x
    prediction = linear_classifier(X, w_svm)
    m = -w_svm[1] / w_svm[2]
    b = -w_svm[0] / w_svm[2]
    axes[i].plot(X[:, 0], m*X[:, 0] + b, ls = '-')
    axes[i].scatter(X[:, 0], X[:, 1], c = clrs[y_blob + 1])
    axes[i].scatter(X[:, 0], X[:, 1], c = clrs[prediction + 1])
    axes[i].set_title("alpha = {}".format(alphas[i]))
```



Find the optimal set of hyperparameters for an SVM model with Dataset 1.

Use `GridSearchCV` and find the optimal value of α and γ.

In [38]:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics.pairwise import rbf_kernel
```

```
np.random.seed(0)

alphas = np.array([1,2,10,100])
Cs = 1./alphas
sigmas = np.array([5, 10, 15, 20])
gammas = 1./(2*sigmas**2)
params = {'C': Cs}
r2s = []
for i in np.array([0,1,2,3]):
    svc_model = SVC(kernel = 'rbf', gamma = gammas[i],C=Cs[i])
    x_train = rbf_kernel(X, X,gammas[i])
    svc_search = GridSearchCV(svc_model,params,cv=3)
    svc_search.fit(x_train,y)
    print('For {}, r2 = : {}, alpha = {}'.format(svc_search.best_estim
    r2s.append(svc_search.best_score_)

#I could not use alpha is 0 because when passing in the C vector I got
print(' ')
```

```
For SVC(gamma=0.02), r2 = : 0.854741444293683, alpha = 1
For SVC(gamma=0.005), r2 = : 0.8649178350670889, alpha = 2
For SVC(gamma=0.0022222222222222222), r2 = : 0.5859339665309814, alph
a = 10
For SVC(gamma=0.00125), r2 = : 0.5808834614804764, alpha = 100

The optimal alpha is 1, and in my case with a gamma of 0.02
```

Calculate the accruacy, precision, and recall for the best model.

You can write your own function that calculates the metrics or you may use built-in functions.

In [39]:

```
def acc_prec_recall(y_model, y_actual):
    TP = np.sum(np.logical_and(y_model == y_actual, y_model == 1))
    TN = np.sum(np.logical_and(y_model == y_actual, y_model == 0))
    FP = np.sum(np.logical_and(y_model != y_actual, y_model == 1))
    FN = np.sum(np.logical_and(y_model != y_actual, y_model == 0))
    acc = (TP + TN) / (TP + TN + FP + FN)
    if TP == 0:
        prec = 0
        recall = 0
    else:
        prec = TP / (TP + FP)
        recall = TP / (TP + FN)
    return acc, prec, recall
# I know from earlier that the optimal alpha is 1 and gamma is 0.02

svc = SVC(kernel = 'rbf',gamma = 0.02, C=1)
svc.fit(X,y)
y_model = svc.predict(X)
ans = acc_prec_recall(y_model,y)
```

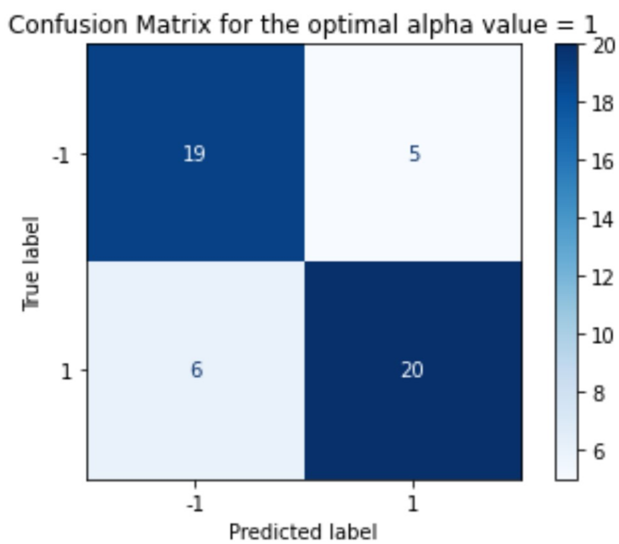
```
(0.8924731182795699, 0.8924731182795699, 1.0)
```

Plot the confusion matrix.

In [40]:

```
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix

# I looked up the documentation from sklearn

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state
svc_model = SVC(kernel='rbf', C=1)
svc_model.fit(X_train, y_train)
disp = plot_confusion_matrix(svc_model, X_test, y_test,cmap=plt.cm.Blu
disp.ax_.set_title('Confusion Matrix for the optimal alpha value = 1')
print(disp.confusion_matrix)
```

```
[[19  5]
 [ 6 20]]
```



What happens to the decision boundary as α goes to ∞?

As alpha goes to infinity C goes to 0, and as C decreases we have more "support vectors," meaning the decision boundary between the different classes is not as precise around the classes.

What happens to the decision boundary as γ goes to 0?

As gamma goes to 0 the boundary is "less complex" which I take to mean the same thing as if the alpha goes to infinity, the decision boundary will be less precise when separating the classes.

4. 6745 Only: Analytical Derivation - I am taking 4745, not 6745! I am not a grad student

Derive an analytical expression for the gradient of the softmax function with respect to $\vec{w}$.

The **softmax** loss function is defined as:

$$g(\vec{w}) = \sum_i log(1 + \exp(-y_i \vec{x}_i^T \vec{w}))$$

where $\vec{x}_i$ is the i-th row of the input matrix $\bar{\bar{X}}$.

Hint 1: The function $g(\vec{w})$ can be expressed as $f(r(s(\vec{w})))$ where r and s are arbitrary functions and the chain rule can be applied.

Type *Markdown* and LaTeX: α^2

Optional: Logistic regression from the regression perspective

An alternate interpretation of classification is that we are performing non-linear regression to fit a **step function** to our data (because the output is whether 0 or 1). Since step functions are not differentiable at the step, a smooth approximation with non-zero derivatives must be used. One such approximation is the *tanh* function:

$$\tanh(x) = \frac{2}{1+\exp(-x)} - 1$$

This leads to a reformulation of the classification problem as:

$$\vec{y} = \tanh(\bar{\bar{X}}\vec{w})$$

Show that this is mathematically equivalent to **logistic regression**, or minimization of the **softmax** cost function.

Type *Markdown* and LaTeX: α^2