worksheet_21

November 27, 2023

1 Worksheet 21

Name: Liang Han UID: U86104920

1.0.1 Topics

- Logistic Regression

1.1 Logistic Regression

[40]:
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets as datasets
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures

centers = [[0, 0]]
t, _ = datasets.make_blobs(n_samples=750, centers=centers, cluster_std=1,␣
↪random_state=0)


# LINE
def generate_line_data():
    # create some space between the classes
    X = np.array(list(filter(lambda x: x[0] - x[1] < -.5 or x[0] - x[1] > .5,␣
↪t)))
    Y = np.array([1 if x[0] - x[1] >= 0 else 0 for x in X])
    return X, Y


# CIRCLE
def generate_circle_data():
    # create some space between the classes
    X = np.array(list(filter(
        lambda x: (x[0] - centers[0][0]) ** 2 + (x[1] - centers[0][1]) ** 2 < 1␣
↪or (x[0] - centers[0][0]) ** 2 + (
                x[1] - centers[0][1]) ** 2 > 1.5, t)))
```

```
    Y = np.array([1 if (x[0] - centers[0][0]) ** 2 + (x[1] - centers[0][1]) **
 ↪2 >= 1 else 0 for x in X])
    return X, Y


# XOR
def generate_xor_data():
    X = np.array([
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1]])
    Y = np.array([x[0] ^ x[1] for x in X])
    return X, Y
```

a) Using the above code, generate and plot data that is linearly separable.

```
[41]: X, Y = generate_line_data()
```

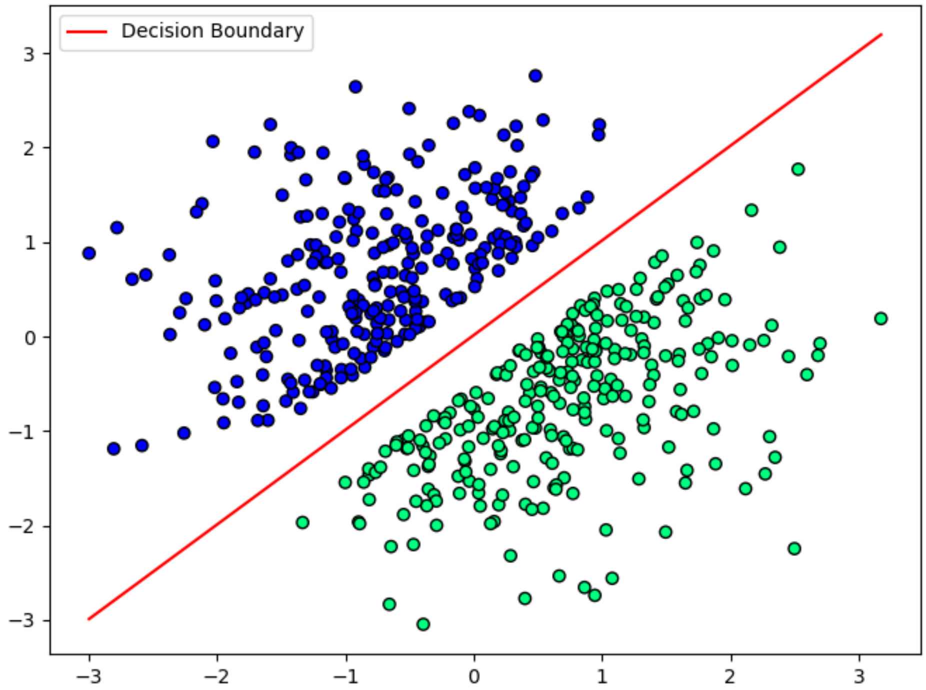
b) Fit a logistic regression model to the data a print out the coefficients.

```
[42]: model = LogisticRegression().fit(X, Y)
      model.coef_
      model.intercept_
```

```
[42]: array([0.05839469])
```

c) Using the coefficients, plot the line through the scatter plot you created in a). (Note: you need to do some math to get the line in the right form)

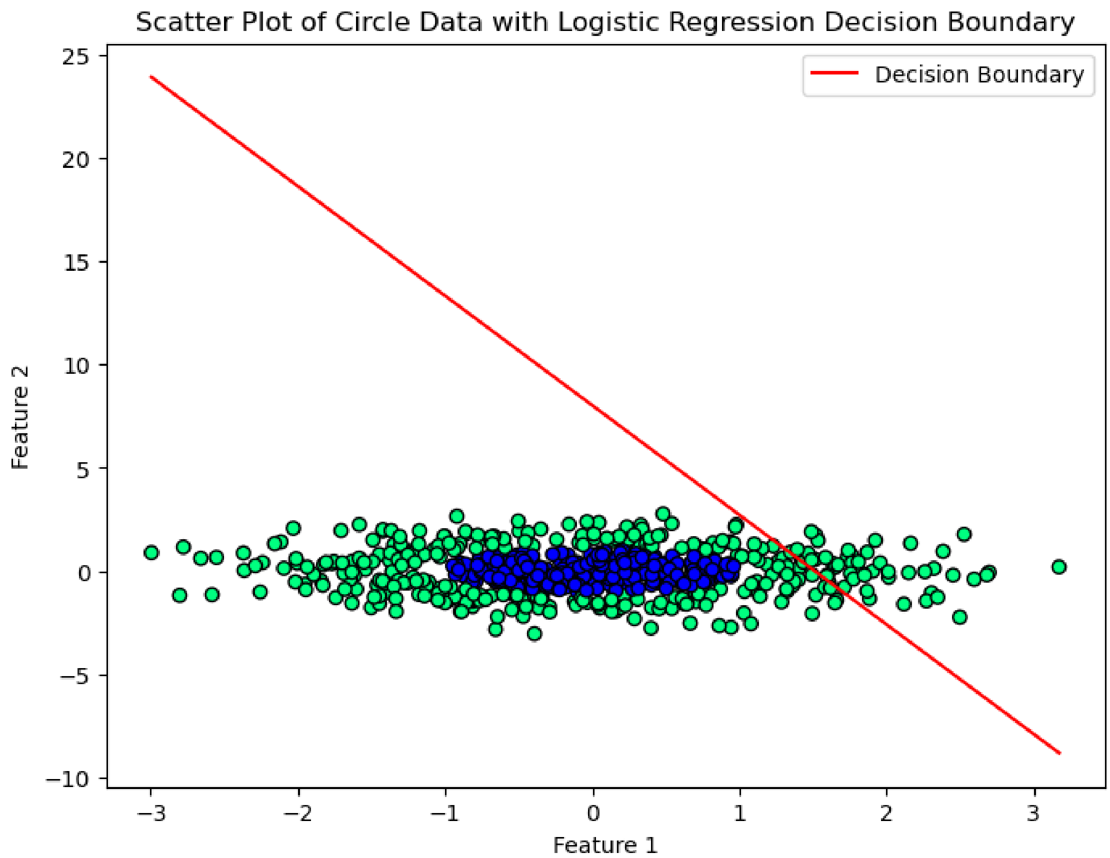
```
[43]: intercept = model.intercept_[0]
      coef = model.coef_[0]

      x_values = np.array([min(X[:, 0]), max(X[:, 0])])
      y_values = -(intercept + coef[0] * x_values) / coef[1]

      plt.figure(figsize=(8, 6))
      plt.scatter(X[:, 0], X[:, 1], c=Y, cmap='winter', edgecolor='k')
      plt.plot(x_values, y_values, label="Decision Boundary", color='red')
      plt.legend()
      plt.show()
```



d) Using the above code, generate and plot the CIRCLE data.

[10]:
```
# Generate the circle data
X_circle, Y_circle = generate_circle_data()

# Fit the logistic regression model to the circle data
model_circle = LogisticRegression().fit(X_circle, Y_circle)

# Extract coefficients and intercept
coef_circle = model_circle.coef_[0]
intercept_circle = model_circle.intercept_[0]

# Calculate the decision boundary line for circle data
# For logistic regression, this isn't a straight line in the case of circular␣
↪data, but we plot it anyway for demonstration
x_values_circle = np.array([min(X_circle[:, 0]), max(X_circle[:, 0])])
y_values_circle = -(intercept_circle + coef_circle[0] * x_values_circle) /␣
↪coef_circle[1]

# Plotting
plt.figure(figsize=(8, 6))
```

```
plt.scatter(X_circle[:, 0], X_circle[:, 1], c=Y_circle, cmap='winter',
    edgecolor='k')
plt.plot(x_values_circle, y_values_circle, label="Decision Boundary",
    color='red')
plt.legend()
plt.show()
t
```



[10]:
```
array([[-0.39944903,  0.37005589],
       [-0.38687085, -0.51029274],
       [-0.80340966, -0.68954978],
       …,
       [-1.83002855, -0.69583512],
       [ 0.28427967,  1.74266878],
       [-0.9988488 , -0.74013679]])
```

e) Notice that the equation of an ellipse is of the form

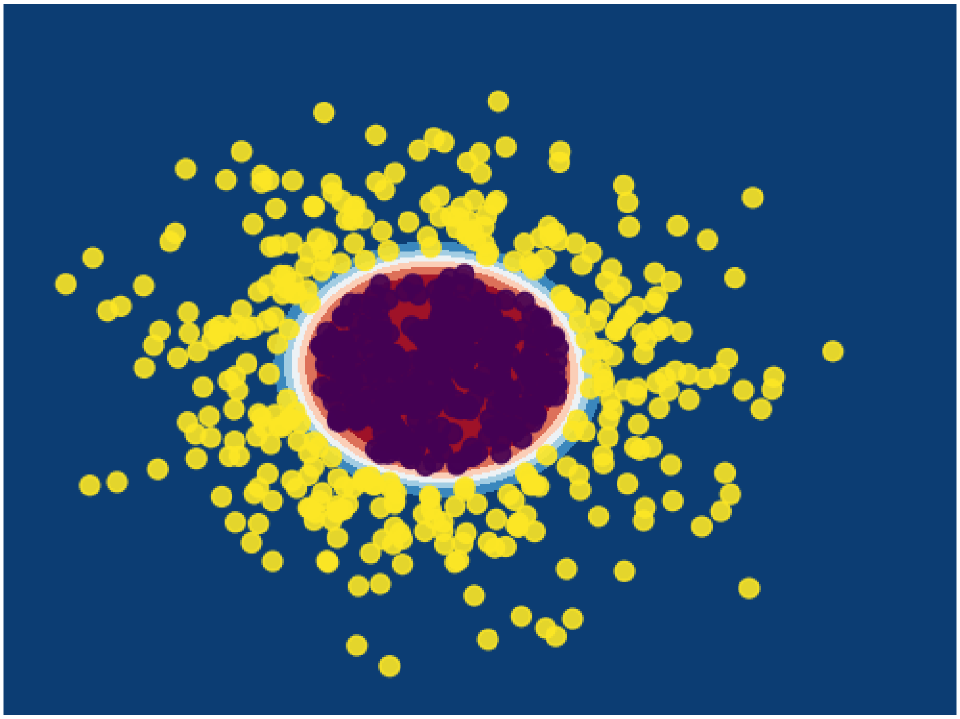
$$ax^2 + by^2 = c$$

Fit a logistic regression model to an appropriate transformation of X.

[16]:
```python
from sklearn.preprocessing import PolynomialFeatures

# Transform the features by squaring them
poly = PolynomialFeatures(degree=2, include_bias=False)
X_transformed = poly.fit_transform(X_circle)

# Fit the logistic regression model to the transformed features
model_transformed = LogisticRegression().fit(X_transformed, Y_circle)
```

f) Plot the decision boundary using the code below.

[17]:
```python
# Creating a mesh to plot in
h = .02  # step size in the mesh
x_min, x_max = X_circle[:, 0].min() - .5, X_circle[:, 0].max() + 1
y_min, y_max = X_circle[:, 1].min() - .5, X_circle[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# Transforming the mesh data for prediction
meshData = np.c_[xx.ravel(), yy.ravel()]
meshData_transformed = poly.transform(meshData)

# Predicting probabilities and labels for the mesh
A = model_transformed.predict_proba(meshData_transformed)[:, 1].reshape(xx.
 ↪shape)
Z = model_transformed.predict(meshData_transformed).reshape(xx.shape)

# Plotting
fig, ax = plt.subplots()
ax.contourf(xx, yy, A, cmap="RdBu", vmin=0, vmax=1)
ax.axis('off')

# Plotting also the training points
ax.scatter(X_circle[:, 0], X_circle[:, 1], c=Y_circle, s=50, alpha=0.9)
plt.show()
```

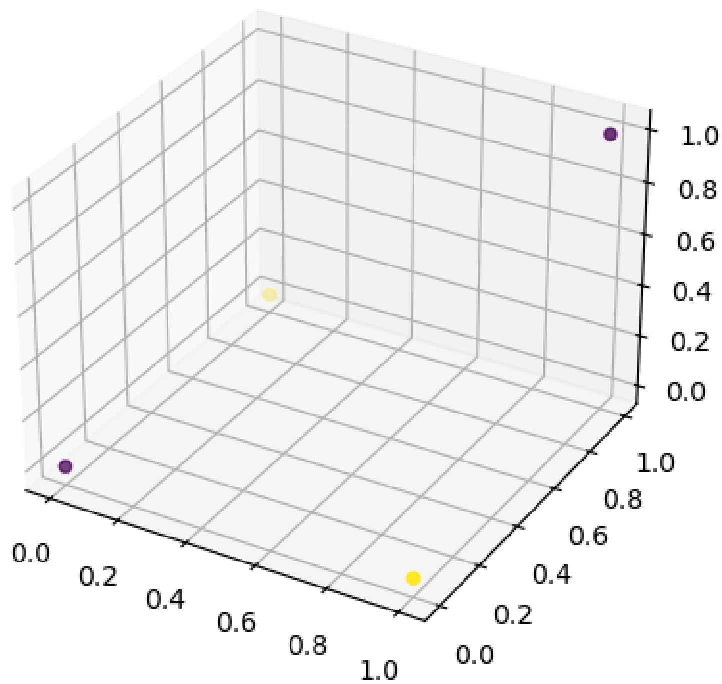


g) Plot the XOR data. In this 2D space, the data is not linearly separable, but by introducing a new feature

$$x_3 = x_1 * x_2$$

(called an interaction term) we should be able to find a hyperplane that separates the data in 3D. Plot this new dataset in 3D.

[18]:
```
from mpl_toolkits.mplot3d import Axes3D

X, Y = generate_xor_data()
ax = plt.axes(projection='3d')
ax.scatter3D(X[:, 0], X[:, 1], X[:, 0] * X[:, 1], c=Y)
plt.show()
```



h) Apply a logistic regression model using the interaction term. Plot the decision boundary.

[20]:
```
poly = PolynomialFeatures(interaction_only=True)
lr = LogisticRegression(verbose=0)
model = make_pipeline(poly, lr).fit(X, Y)

# create a mesh to plot in
h = .02  # step size in the mesh
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
meshData = np.c_[xx.ravel(), yy.ravel()]

fig, ax = plt.subplots()
A = model.predict_proba(meshData)[:, 1].reshape(xx.shape)
Z = model.predict(meshData).reshape(xx.shape)
ax.contourf(xx, yy, A, cmap="RdBu", vmin=0, vmax=1)
ax.axis('off')

# Plot also the training points
ax.scatter(X[:, 0], X[:, 1], color=Y, s=50, alpha=0.9)
plt.show()
```

[44]:
```
%matplotlib widget
for i in range(20000):
    for solver in ['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag',␣
 ↪'saga']:
        X_transform = PolynomialFeatures(interaction_only=True,␣
 ↪include_bias=False).fit_transform(X)
        model = LogisticRegression(verbose=0, solver=solver, random_state=i,␣
 ↪max_iter=10000)
        model.fit(X_transform, Y)
        # print(model.score(X_transform, Y))
        if model.score(X_transform, Y) > .75:
            # print("random state = ", i)
            # print("solver = ", solver)
            break

# print(model.coef_)
# print(model.intercept_)

xx, yy = np.meshgrid([x / 10 for x in range(-1, 11)], [x / 10 for x in␣
 ↪range(-1, 11)])
z = - model.intercept_ / model.coef_[0][2] - model.coef_[0][0] * xx / model.
 ↪coef_[0][2] - model.coef_[0][1] * yy / \
    model.coef_[0][2]
```

```
ax = plt.axes(projection='3d')
ax.scatter3D(X[:, 0], X[:, 1], X[:, 0] * X[:, 1], c=Y)
ax.plot_surface(xx, yy, z, alpha=0.5)
plt.show()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[44], line 6
      4 X_transform = PolynomialFeatures(interaction_only=True,␣
 ↪include_bias=False).fit_transform(X)
      5 model = LogisticRegression(verbose=0, solver=solver, random_state=i,␣
 ↪max_iter=10000)
----> 6 model.fit(X_transform, Y)
      7 # print(model.score(X_transform, Y))
      8 if model.score(X_transform, Y) > .75:
      9     # print("random state = ", i)
     10     # print("solver = ", solver)

File ~\anaconda3\envs\py11\Lib\site-packages\sklearn\base.py:1152, in␣
 ↪_fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1145     estimator._validate_params()
   1147 with config_context(
   1148     skip_parameter_validation=(
   1149         prefer_skip_nested_validation or global_skip_validation
   1150     )
   1151 ):
-> 1152     return fit_method(estimator, *args, **kwargs)

File ~\anaconda3\envs\py11\Lib\site-packages\sklearn\linear_model\_logistic.py:
 ↪1303, in LogisticRegression.fit(self, X, y, sample_weight)
   1300 else:
   1301     n_threads = 1
-> 1303 fold_coefs_ =␣
 ↪Parallel(n_jobs=self.n_jobs, verbose=self.verbose, prefer=prefer)(
   1304     path_func(
   1305         X,
   1306         y,
   1307         pos_class=class_,
   1308         Cs=[C_],
   1309         l1_ratio=self.l1_ratio,
   1310         fit_intercept=self.fit_intercept,
   1311         tol=self.tol,
   1312         verbose=self.verbose,
   1313         solver=solver,
   1314         multi_class=multi_class,
   1315         max_iter=self.max_iter,
```

```
   1316         class_weight=self.class_weight,
   1317         check_input=False,
   1318         random_state=self.random_state,
   1319         coef=warm_start_coef_,
   1320         penalty=penalty,
   1321         max_squared_sum=max_squared_sum,
   1322         sample_weight=sample_weight,
   1323         n_threads=n_threads,
   1324     )
   1325     for class_, warm_start_coef_ in zip(classes_, warm_start_coef)
   1326 )
   1328 fold_coefs_, _, n_iter_ = zip(*fold_coefs_)
   1329 self.n_iter_ = np.asarray(n_iter_, dtype=np.int32)[:, 0]

File ~\anaconda3\envs\py11\Lib\site-packages\sklearn\utils\parallel.py:65, in Parallel.__call__(self, iterable)
     60 config = get_config()
     61 iterable_with_config = (
     62     (_with_config(delayed_func, config), args, kwargs)
     63     for delayed_func, args, kwargs in iterable
     64 )
---> 65 return super().__call__(iterable_with_config)

File ~\anaconda3\envs\py11\Lib\site-packages\joblib\parallel.py:1863, in Parallel.__call__(self, iterable)
   1861     output = self._get_sequential_output(iterable)
   1862     next(output)
-> 1863     return output if self.return_generator else list(output)
   1865 # Let's create an ID that uniquely identifies the current call. If the
   1866 # call is interrupted early and that the same instance is immediately
   1867 # re-used, this id will be used to prevent workers that were
   1868 # concurrently finalizing a task from the previous call to run the
   1869 # callback.
   1870 with self._lock:

File ~\anaconda3\envs\py11\Lib\site-packages\joblib\parallel.py:1792, in Parallel._get_sequential_output(self, iterable)
   1790 self.n_dispatched_batches += 1
   1791 self.n_dispatched_tasks += 1
-> 1792 res = func(*args, **kwargs)
   1793 self.n_completed_tasks += 1
   1794 self.print_progress()

File ~\anaconda3\envs\py11\Lib\site-packages\sklearn\utils\parallel.py:127, in _FuncWrapper.__call__(self, *args, **kwargs)
    125     config = {}
    126 with config_context(**config):
--> 127     return self.function(*args, **kwargs)
```

```
File ~\anaconda3\envs\py11\Lib\site-packages\sklearn\linear_model\_logistic.py:452, in _logistic_regression_path(X, y, pos_class, Cs, fit_intercept, max_iter, tol, verbose, solver, coef, class_weight, dual, penalty, intercept_scaling, multi_class, random_state, check_input, max_squared_sum, sample_weight, l1_ratio, n_threads)
    448 l2_reg_strength = 1.0 / C
    449 iprint = [-1, 50, 1, 100, 101][
    450     np.searchsorted(np.array([0, 1, 2, 3]), verbose)
    451 ]
--> 452 opt_res = optimize.minimize(
    453     func,
    454     w0,
    455     method="L-BFGS-B",
    456     jac=True,
    457     args=(X, target, sample_weight, l2_reg_strength, n_threads),
    458     options={"iprint": iprint, "gtol": tol, "maxiter": max_iter},
    459 )
    460 n_iter_i = _check_optimize_result(
    461     solver,
    462     opt_res,
    463     max_iter,
    464     extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
    465 )
    466 w0, loss = opt_res.x, opt_res.fun

File ~\anaconda3\envs\py11\Lib\site-packages\scipy\optimize\_minimize.py:710, i minimize(fun, x0, args, method, jac, hess, hessp, bounds, constraints, tol, callback, options)
    707     res = _minimize_newtoncg(fun, x0, args, jac, hess, hessp, callback,
    708                              **options)
    709 elif meth == 'l-bfgs-b':
--> 710     res = _minimize_lbfgsb(fun, x0, args, jac, bounds,
    711                            callback=callback, **options)
    712 elif meth == 'tnc':
    713     res = _minimize_tnc(fun, x0, args, jac, bounds, callback=callback,
    714                         **options)

File ~\anaconda3\envs\py11\Lib\site-packages\scipy\optimize\_lbfgsb_py.py:365, in _minimize_lbfgsb(fun, x0, args, jac, bounds, disp, maxcor, ftol, gtol, eps, maxfun, maxiter, iprint, callback, maxls, finite_diff_rel_step, **unknown_options)
    359 task_str = task.tobytes()
    360 if task_str.startswith(b'FG'):
    361     # The minimization routine wants f and g at the current x.
    362     # Note that interruptions due to maxfun are postponed
    363     # until the completion of the current minimization iteration.
    364     # Overwrite f and g:
--> 365     f, g = func_and_grad(x)
```

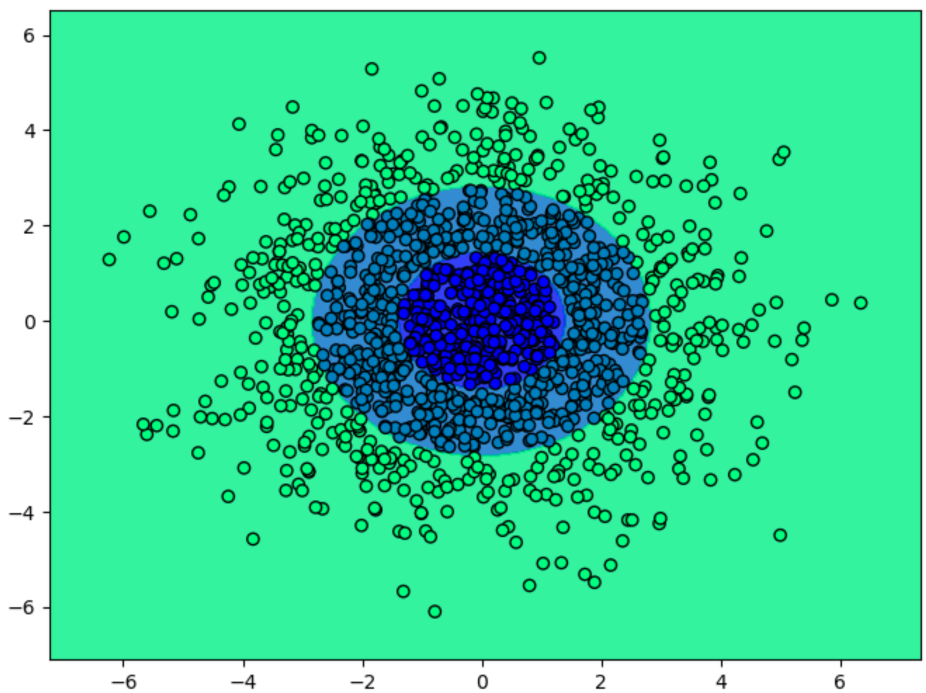
```
    366 elif task_str.startswith(b'NEW_X'):
    367     # new iteration
    368     n_iterations += 1

File ~\anaconda3\envs\py11\Lib\site-packages\scipy\optimize\_differentiable_functions.py:285, in ScalarFunction.fun_and_grad(self, x)
    283 if not np.array_equal(x, self.x):
    284     self._update_x_impl(x)
--> 285 self._update_fun()
    286 self._update_grad()
    287 return self.f, self.g

File ~\anaconda3\envs\py11\Lib\site-packages\scipy\optimize\_differentiable_functions.py:251, in ScalarFunction._update_fun(self)
    249 def _update_fun(self):
    250     if not self.f_updated:
--> 251         self._update_fun_impl()
    252         self.f_updated = True

File ~\anaconda3\envs\py11\Lib\site-packages\scipy\optimize\_differentiable_functions.py:155, in ScalarFunction.__init__.<locals>.update_fun()
    154 def update_fun():
--> 155     self.f = fun_wrapped(self.x)

File ~\anaconda3\envs\py11\Lib\site-packages\scipy\optimize\_differentiable_functions.py:137, in ScalarFunction.__init__.<locals>.fun_wrapped(x)
    133 self.nfev += 1
    134 # Send a copy because the user may overwrite it.
    135 # Overwriting results in undefined behaviour because
    136 # fun(self.x) will change self.x, with the two no longer linked.
--> 137 fx = fun(np.copy(x), *args)
    138 # Make sure the function returns a true scalar
    139 if not np.isscalar(fx):

File ~\anaconda3\envs\py11\Lib\site-packages\scipy\optimize\_optimize.py:77, in MemoizeJac.__call__(self, x, *args)
     75 def __call__(self, x, *args):
     76     """ returns the function value """
---> 77     self._compute_if_needed(x, *args)
     78     return self._value

File ~\anaconda3\envs\py11\Lib\site-packages\scipy\optimize\_optimize.py:71, in MemoizeJac._compute_if_needed(self, x, *args)
     69 if not np.all(x == self.x) or self._value is None or self.jac is None:
     70     self.x = np.asarray(x).copy()
```

```
---> 71      fg = self.fun(x, *args)
     72      self.jac = fg[1]
     73      self._value = fg[0]

File ~\anaconda3\envs\py11\Lib\site-packages\sklearn\linear_model\_linear_loss.
↪py:279, in LinearModelLoss.loss_gradient(self, coef, X, y, sample_weight,␣
↪l2_reg_strength, n_threads, raw_prediction)
    276 else:
    277     weights, intercept = self.weight_intercept(coef)
--> 279 loss, grad_pointwise = self.base_loss.loss_gradient(
    280     y_true=y,
    281     raw_prediction=raw_prediction,
    282     sample_weight=sample_weight,
    283     n_threads=n_threads,
    284 )
    285 loss = loss.sum()
    286 loss += self.l2_penalty(weights, l2_reg_strength)

File ~\anaconda3\envs\py11\Lib\site-packages\sklearn\_loss\loss.py:253, in␣
↪BaseLoss.loss_gradient(self, y_true, raw_prediction, sample_weight, loss_out,␣
↪gradient_out, n_threads)
    250 if gradient_out.ndim == 2 and gradient_out.shape[1] == 1:
    251     gradient_out = gradient_out.squeeze(1)
--> 253 return self.closs.loss_gradient(
    254     y_true=y_true,
    255     raw_prediction=raw_prediction,
    256     sample_weight=sample_weight,
    257     loss_out=loss_out,
    258     gradient_out=gradient_out,
    259     n_threads=n_threads,
    260 )

KeyboardInterrupt:
```

i) Using the code below that generates 3 concentric circles, fit a logisitc regression model to it and plot the decision boundary.

[39]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline


def generate_circles_data(t):
    def label(x):
```

```python
        if x[0] ** 2 + x[1] ** 2 >= 2 and x[0] ** 2 + x[1] ** 2 < 8:
            return 1
        if x[0] ** 2 + x[1] ** 2 >= 8:
            return 2
        return 0

    # create some space between the classes
    X = np.array(list(filter(lambda x: (x[0] ** 2 + x[1] ** 2 < 1.8 or x[0] ** 2 + x[1] ** 2 > 2.2) and (
            x[0] ** 2 + x[1] ** 2 < 7.8 or x[0] ** 2 + x[1] ** 2 > 8.2), t)))
    Y = np.array([label(x) for x in X])
    return X, Y


# Generating data
centers = [[0, 0]]
t, _ = make_blobs(n_samples=1500, centers=centers, cluster_std=2, random_state=0)
X_concentric, Y_concentric = generate_circles_data(t)

# Preparing the model with polynomial features and logistic regression
poly_concentric = PolynomialFeatures(degree=2)
lr_concentric = LogisticRegression(max_iter=1000)
model_concentric = make_pipeline(poly_concentric, lr_concentric)

# Fitting the model
model_concentric.fit(X_concentric, Y_concentric)

# Creating a meshgrid for prediction
x_min, x_max = X_concentric[:, 0].min() - 1, X_concentric[:, 0].max() + 1
y_min, y_max = X_concentric[:, 1].min() - 1, X_concentric[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 500), np.linspace(y_min, y_max, 500))
meshData = np.c_[xx.ravel(), yy.ravel()]

# Predicting the labels for each point in the mesh
Z_concentric = model_concentric.predict(meshData).reshape(xx.shape)

# Plotting
plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z_concentric, alpha=0.8, cmap='winter')
plt.scatter(X_concentric[:, 0], X_concentric[:, 1], c=Y_concentric, edgecolor='k', cmap='winter')
plt.show()
```


6
4
2
0
−2
−4
−6
−6 −4 −2 0 2 4 6