In [ ]:

```
# Initialize OK
from client.api.notebook import Notebook
ok = Notebook('lab11.ok')
```

Lab 11: Logistic Regression

In this lab, you will get practice fitting a logistic regression model on NBA data.

Due Date

This assignment is due on **Monday, November 4 at 11:59pm**.

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about this assignment, we ask that you **write your solutions individually**. If you discuss the assignment with others, please **include their names** in the cell below.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

plt.rcParams['figure.figsize'] = (4, 4)
plt.rcParams['figure.dpi'] = 150
plt.rcParams['lines.linewidth'] = 3
sns.set()
```

In [2]:

```
df = pd.read_csv('nba.csv')
df["WON"] = df["WL"]
df["WON"] = df["WON"].replace("W", 1)
df["WON"] = df["WON"].replace("L", 0)
df.head(5)
```

Question 1: 1D Logistic Regression Model

In lecture we buit a logistic regression classifier for the NBA data loaded above. Specifically, our model took an observation x and a parameter vector β and used them to generate a prediction $\hat{y}$. Note that in this question we will assume that x is a one-dimensional scalar.

In this case, our predictions represented the probability that the observation belonged to a specific category. In lecture the category was whether or not the team won. That is, $\hat{y} = P(Y = 1|x)$, where $Y = 1$ indicates that the team we're observing won the game.

Question 1a: Implementing a 1D Logistic Model

As discussed in lecture, the prediction of our model is $\hat{y} = \sigma(x\hat{\beta})$. *Note: Here, both x and $\hat{\beta}$ are scalars, not vectors.*

In this lab we'll start by trying to build a model that predicts the winning probability as a function of the number of points that a team scored.

Below, first define `sigma` to be the sigmoid function we saw in lecture. Then, fill in `predicted_probability_of_winning_given_pts` so that it returns the correct prediction. Your function should work for both scalar and array arguments for `pts`. That is, `predicted_probability_of_winning_given_pts(100, 0.01)` should return a single value (0.731) and `predicted_probability_of_winning_given_pts(np.array([100, 110])), 0.01)` should return an array of values (0.731, 0.750).

In [3]:

```
def sigma(t):
    # BEGIN SOLUTION
    return 1 / (1 + np.exp(-t))
    # END SOLUTION

def predicted_probability_of_winning_given_pts(pts, beta):
    # BEGIN SOLUTION
    t = pts * beta
    return sigma(t)
    # END SOLUTION
```

In [ ]:

```
ok.grade("q1a");
```

Exploring an Example Model

Suppose we pick $\hat{\beta}$ = 0.01. We can generate predictions for each of the games in our real world dataset as follows:

In [7]:

```
beta = 0.01
x = df["PTS"]
y_obs = df["WON"]
y_hat = predicted_probability_of_winning_given_pts(x, beta)
```

If we look at our predictions, we see that every team is given a greater than 50 percent prediction of winning based on their number of points. This suggests a problem with our model.

In [8]:

```
y_hat.mean()
```

In [9]:

```
plt.plot(y_hat);
```

To understand what's going on, we make a plot of the prediction our model will make as a function of the number of points scored for $\hat{\beta} = 0.01$.

In [10]:

```
beta = 0.01
pts = np.linspace(0, 140, 140)
plt.plot(pts, predicted_probability_of_winning_given_pts(pts, be
ta), color = 'orange')
plt.ylabel("Win Probability")
plt.xlabel("Points Scored");
```

We can also provide the actual results from the NBA dataset as blue stars for comparison to our model.

In [11]:

```
beta = 0.01
pts = np.linspace(0, 140, 140)
plt.plot(pts, predicted_probability_of_winning_given_pts(pts, be
ta), color = 'orange')
plt.plot(df[["PTS"]], df[["WON"]], 'b*')
plt.ylabel("Win Probability")
plt.xlabel("Points")
plt.legend(['$\hat{y}$', 'y']);
```

Question 1b

Is this model reasonable? Why or why not?

SOLUTION: No, no matter a team's score, it seems to always have a greater than 50% chance of winning.

Question 1c

Try playing around with other beta values. You should observe that the models are all pretty bad, no matter what β you pick. Explain why below.

SOLUTION: No matter what beta we pick, the win probability is always at least 0.5.

Question 2: Adding an Intercept Term

If you observe your plot(s) from the previous part of this lab, you'll see that the chance of winning is always at least 0.5 under our model. This is unreasonable, e.g. suppose a team somehow scored only 36 points, they'd have no chance of winning in an NBA game.

Implementing Multiple Linear Regression

To deal with this, we should add another feature to our model. Specifically, we'll add a bias term, i.e. a feature that is equal to 1 for all observations. We've done this for you below.

In [12]:

```
points_and_bias = df[["PTS"]].copy()
points_and_bias["bias"] = np.ones(len(points_and_bias))
points_and_bias.head()
```

Logistic regression generalizes to multiple features in exactly the same manner as linear regression.

Recall that whereas linear regression on one parameter gave predictions $\hat{y} = x\hat{\beta}$, multiple linear regression gave predictions $\hat{y} = \vec{x} \cdot \vec{\hat{\beta}} = \vec{x}^T \vec{\hat{\beta}} = \sum_{i=1}^{p} x_i \hat{\beta}_i$.

Logistic regression generalizes in exactly the same way. That is logistic regression in 1 variable is given by $\hat{y} = \sigma(x\hat{\beta})$, whereas multiple logistic regression is given by $\hat{y} = \sigma(\vec{x} \cdot \vec{\hat{\beta}}) = \sigma(\vec{x}^T \vec{\hat{\beta}}) = \sigma(\sum_{i=1}^{p} x_i \hat{\beta}_i)$.

Fill in the function below so that it returns predictions as described above. As in question 1, your model should be able to handle scalar and array arguments for x. For example `predicted_probability_of_winning_given_features(X.iloc[0:3, :], [0.1, -10])` should return a list (or series) (or numpy array) of the values `[0.6899744811276126, 0.5, 0.21416501695744153]`.

Your function only needs to work for array inputs to `x`. That is, your code does not need to work properly for `predicted_probability_of_winning_given_features(110, [0.1, -10])`

In [13]:

```
def predicted_probability_of_winning_given_features(X, beta):
    # BEGIN SOLUTION
    t = X @ beta
    return sigma(t)
    # END SOLUTION
```

In [ ]:

```
ok.grade("q2a");
```

Exploring Multiple Logistic Regression

Now we have two parameters β_1 and β_2. Suppose $\beta_1 = 0.001$ and $\beta_2 = 2$. We can compute the predicted probability that each team won during each game as follows.

In [17]:

```
beta = np.array([0.001, 2])
predicted_probability_of_winning_given_features(points_and_bias.
iloc[0:3, :], beta)
```

In [18]:

```
points_and_bias.iloc[0:3, :]
```

Now that we have a bias term, we have more freedom to adjust our model.

For example, if $\beta_1 = 0.05$ and $\beta_2 = -5$, we get the curve below. Here, the prediction of your model is $\sigma(\beta_1 \times PTS + \beta_2)$. That is, β_1 is the weight of `PTS`, and β_2 is the weight of the bias term.

In [19]:

```
beta = [0.05, -5]
pts = np.linspace(50, 160, 111).reshape(-1, 1)
bias = np.ones(len(pts)).reshape(-1, 1)
point_range_and_bias = np.hstack((pts, bias))
plt.plot(pts, predicted_probability_of_winning_given_features(po
int_range_and_bias, beta), 'orange')
plt.ylabel("Win Probability")
plt.xlabel("Points");
```

And as before, we can also plot the actual data from our NBA dataset for comparison with our model.

In [20]:

```
plt.plot(df["PTS"], df["WON"], 'b*')
beta = [0.05, -5]
pts = np.linspace(50, 160, 111).reshape(-1, 1)
bias = np.ones(len(pts)).reshape(-1, 1)
point_range_and_bias = np.hstack((pts, bias))
plt.plot(pts, predicted_probability_of_winning_given_features(po
int_range_and_bias, beta), 'orange')
plt.ylabel("Win Probability")
plt.xlabel("Points")
plt.legend(['$\hat{y}$', 'y']);
```

Question 2b

Using the plot above, try adjusting β_2 (only). Describe how changing β_2 affects the prediction curve. Provide your description in the cell below.

SOLUTION: Increasing β_2 shifts the curve to the left.

Question 2c

Now using the plot below try adjusting β_1 and β_2 such that you get a sharp curve that is centered at 100 points. In the cell below `beta` should be a list with your chosen values of β_1 and β_2.

- By "centered at 100 points", we mean that $\hat{y}$ should be equal to 0.5 when $x = 100$.
- By "sharp", we mean that the probability should be less than 5% percent for $x = 80$, and greater than 95% for $x = 100$.
- *Hint*: $\sigma(t) = 0.5$ when $t = 0$.

In [21]:

```
plt.plot(df["PTS"], df["WON"], 'b*')
beta = [0.2, -20] # SOLUTION
pts = np.linspace(50, 160, 111).reshape(-1, 1)
bias = np.ones(len(pts)).reshape(-1, 1)
point_range_and_bias = np.hstack((pts, bias))
plt.plot(pts, predicted_probability_of_winning_given_features(po
int_range_and_bias, beta), 'orange')
plt.ylabel("Win Probability")
plt.xlabel("Points")
plt.legend(['$\hat{y}$', 'y']);
```

Provide your β_1 and β_2 in the cell below.

In [22]:

```
beta1 = 0.2 # SOLUTION
beta2 = -20 # SOLUTION
```

In [ ]:

```
ok.grade("q2c");
```

Question 3 : Optimizing Multiple Logistic Regresion

Let's now work towards finding the optimal beta $\hat{\vec{\beta}}$ for our given data.

Note: In the previous question, we referred to our βs without a hat, since we had yet to find the optimal values of $\vec{\beta}$ procedurally.

Question 3a: Calculating MSE

Create a function `mse_for_model_on_NBA_data(beta)` that takes in a value of $\vec{\beta}$ and returns the MSE on the dataset from above. You will first need to define the function `mse(y_obs, y_hat)` which finds the mean squared error between `y_obs` and `y_hat`.

Hint: You need to compute $\hat{y}$ using the given $\vec{\beta}$, then the mean squared error between $\hat{y}$ and the observed data y.

Hint: Use `points_and_bias` and `df["WON"]`.

In [24]:

```
def mse(y_obs, y_hat):
    return np.mean((y_obs - y_hat)**2) # SOLUTION

def mse_for_model_on_NBA_data(beta):
    # BEGIN SOLUTION
    y_hat = predicted_probability_of_winning_given_features(points_and_bias, beta)
    y_obs = df["WON"]
    return mse(y_obs, y_hat)
    # END SOLUTION
```

In [ ]:

```
ok.grade("q3a");
```

Plotting MSE

The cell below plots your MSE function. We're providing this plot purely for your edification. Warning: This code can be pretty slow and might take a minute or two to run.

Note that the surface has a huge almost completely flat region. This means this loss function is very difficult to optimize.

In [26]:

```
import plotly.graph_objects as go

num_points = 50 # increase for better resolution, but it will ru
n more slowly.

if (num_points <= 100):

    uvalues = np.linspace(-0.3, 0.3, num_points)
    vvalues = np.linspace(-20, 20, num_points)
    (u,v) = np.meshgrid(uvalues, vvalues)
    thetas = np.vstack((u.flatten(),v.flatten()))

    MSE = np.array([mse_for_model_on_NBA_data(t) for t in thetas
.T])

    loss_surface = go.Surface(x=u, y=v, z=np.reshape(MSE, u.shap
e))

    fig = go.Figure(data=[loss_surface])
    fig.update_layout(scene = dict(
        xaxis_title = "theta0",
        yaxis_title = "theta1",
        zaxis_title = "MSE"))
    fig.show()
else:
    print("Picking num points > 100 can be really slow. If you r
eally want to try, edit the code above so that this if statement
doesn't trigger.")
```

Question 3b: Minimizing MSE

Using `scipy.optimize.minimize` , find the optimal $\vec{\hat{\beta}}$. Give your answer as `beta_hat_1` and `beta_hat_2` . The resulting MSE should be less than 0.2.

Note: Your starting guess should be (0, 0). If you start somewhere over in the flat region like (0, 20), then scipy.optimize.minimize will get stuck.

Note: The test for this question requires that you did Question 3a correctly.

In [27]:

```
from scipy.optimize import minimize

optimal_beta = minimize(mse_for_model_on_NBA_data, x0 = [0, 0])[
'x'] # SOLUTION
beta_hat_1 = optimal_beta[0] # SOLUTION
beta_hat_2 = optimal_beta[1] # SOLUTION
```

In [ ]:

```
ok.grade("q3b");
```

Question 3c

Finally, let's try to understand how our model can be practically useful. As we'll see in lecture on 11/5, we often convert our logistic regression models into a concrete prediction by thresholding. That is, if our $\hat{y} \geq 0.5$, we say our prediction is that the team will win; otherwise, we say that we predict that we will lose. A simple way to do this is just to round our $\hat{y}$.

In [29]:

```
y_hat = predicted_probability_of_winning_given_features(points_a
nd_bias, np.array([beta_hat_1, beta_hat_2]))
games_and_predictions = df.copy()
games_and_predictions["predicted_to_win"] = np.round(y_hat)
games_and_predictions[["TEAM_NAME", "GAME_DATE", "WON", "predict
ed_to_win"]].tail(5)
```

To evaluate the quality of your model, compute the fraction of the rows of the table for which your model was able to correctly predict the outcome of the game based on only the points scored by one team. Assign this to the variable `percentage_correct`.

In [30]:

```
percentage_correct = ...
# BEGIN SOLUTION NO PROMPT
correct = sum(games_and_predictions['WON'] == games_and_predicti
ons["predicted_to_win"])
total = len(games_and_predictions)
percentage_correct = correct / total
# END SOLUTION
```

In [ ]:

```
ok.grade("q3c");
```

Question 3d

Recall that the surface for the MSE has a huge almost completely flat region, which means that the loss function is very difficult to optimize.

In lecture on Tuesday 11/5, we'll talk about an alternate loss function called the cross-entropy loss that will yield a much nicer loss surface (no big flat regions). We have defined `cel(y_obs, y_hat)` for you; this function calculates the cross-entropy loss betweeen `y_obs` and `y_hat`. Create a function `cel_for_model_on_NBA_data(beta)` that takes in a value of $\vec{\beta}$ and returns the cross-entropy loss on the dataset from question 2.

Hint: Your code for this part should be very similar to your code for question 3a.

In [32]:

```
def cel(y_obs, y_hat):
    return -np.mean(y_obs * np.log(y_hat) + (1 - y_obs) * np.log
(1 - y_hat))

def cel_for_model_on_NBA_data(beta):
    # BEGIN SOLUTION
    y_hat = predicted_probability_of_winning_given_features(poin
ts_and_bias, beta)
    y_obs = df["WON"]
    return cel(y_obs, y_hat)
    # END SOLUTION
```

In [ ]:

```
ok.grade("q3d");
```

Plotting Cross-Entropy Loss

The cell below plots your cross-entropy loss function. Note that the surface has no big flat regions, which makes it easy to optimize.

Note: Feel free to ignore the divide by zero warning.

In [34]:

```
import plotly.graph_objects as go

num_points = 50 # increase for better resolution, but it will ru
n more slowly.

if (num_points <= 100):

    uvalues = np.linspace(-0.3, 0.3, num_points)
    vvalues = np.linspace(-20, 20, num_points)
    (u,v) = np.meshgrid(uvalues, vvalues)
    thetas = np.vstack((u.flatten(),v.flatten()))

    CEL = np.array([cel_for_model_on_NBA_data(t) for t in thetas
.T])

    loss_surface = go.Surface(x=u, y=v, z=np.reshape(CEL, u.shap
e))

    fig = go.Figure(data=[loss_surface])
    fig.update_layout(scene = dict(
        xaxis_title = "theta0",
        yaxis_title = "theta1",
        zaxis_title = "CEL"))
    fig.show()
else:
    print("Picking num points > 100 can be really slow. If you r
eally want to try, edit the code above so that this if statement
doesn't trigger.")
```

In this lab we only fit a logistic regression model one one feature (the number of points) and an intercept term. However, we can easily train a higher dimensional logistic regression model by using more features that provide useful information. For example, we can add the column `FGM` to our model in addition to `PTS`. Note that this is similar to the difference between simple linear regression and multiple linear regression. As an optional exercise, consider adding more features to your logistic regression model to increase `percentage_correct` from question 3c.

Make sure to complete Vitamin 11 on Gradescope by 11:59 PM on Monday, 11/4!

Submission

Congratulations! You are finished with this assignment. Please don't forget to submit by 11:59pm on Monday, 11/4!

Submit

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. **Please save before submitting!**

In [ ]:

```
# Save your notebook first, then run this cell to submit.
ok.submit()
```