Wasil Engel
12231558
PSet 2

I partnered up with Ricardo Saucedo (12245077).

To see more than the screenshots of the plots I created for the coding part of the assignment, please see my Jupyter notebook on Github here: https://github.com/Wasil-UChi/Machine-Learning

Ch 4 CLASSIFICATION

 # 6 **p.170**

Ch 4, no 6   Logistic regression → logit ?

a) $p(Y) = -6 + 0.05\,\text{hours} + 1\,\text{gpa}$

→ Fit this in logit function: $\frac{\exp(Y)}{1+\exp(Y)} = \frac{\exp(-6+0.05(40)+1(3.5))}{1+\exp(-6+0.05(40)+1(3.5))}$

$$= \frac{\exp(-0.5)}{1+\exp(-0.5)} = \frac{0.6065}{1+0.6065} = 0.37751$$

Such a student has a probability of 37.75% to end up with an A.

b) $p(Y) = \frac{\exp(-6+0.05\,\text{hours}+1(3.5))}{1+\exp(-6+0.05\,\text{hours}+1(3.5))} = 0.5$ → solve for hours

$$\exp(-2.5+0.05\,\text{hours}) = 0.5\,(1+\exp(-2.5+0.05\,\text{hours}))$$

$$1\exp(-2.5+0.05\,\text{hours}) = 1/2 + 1/2\exp(-2.5+0.05\,\text{hours})$$

$$1/2\exp(-2.5+0.05\,\text{hours}) = 1/2$$ || use log to get rid off exp!

$$\log(1/2) + (-2.5+0.05\,\text{hours}) = \log(1/2)$$ || multiplication → addition

$$-0.301 + (-2.5+0.05\,\text{hours}) = -0.301$$

$$-2.5 + 0.05\,\text{hours} = 0$$

$$0.05\,\text{hours} = 2.5$$

$$\text{hours} = 50$$

It'd take the student 50 hours to have a 50% chance to get an A.

ch 4, no 7 Bayes' theorem:

$\pi_{yes} = 0.8 \Rightarrow \pi_{no} = 0.2$
$\sigma^2 = 36$
$\bar{X}_{yes} = 10, \bar{X}_{no} = 0$

$$Pr(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$ || Rewrite as:

$$Pr(Y=k \mid X=x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{k} \pi_l f_l(x)}$$ || plug in density function for f(x)

$$Pr(y=yes \mid X=4) = \frac{\pi_{yes}\left(\frac{e^{-\frac{(4-\bar{x}_{yes})^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}\right)}{\pi_{yes}\left(\frac{e^{-\frac{(4-\bar{x}_{yes})^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}\right) + \pi_{no}\left(\frac{e^{-\frac{(4-\bar{x}_{no})^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}\right)}$$

↳ company issues dividend ↳ last year's %age profit

$$= \frac{0.8\left(\frac{e^{-\frac{(4-10)^2}{2(36)}}}{\sqrt{2\pi(36)}}\right)}{0.8\left(\frac{e^{-\frac{(4-10)^2}{2(36)}}}{\sqrt{2(\pi)(36)}}\right) + 0.2\left(\frac{e^{-\frac{(4-0)^2}{2(36)}}}{\sqrt{2(\pi)(36)}}\right)}$$

$$= \frac{0.8\left(\frac{e^{-\frac{36}{72}}}{\sqrt{226.08}}\right)}{0.8\left(\frac{e^{-\frac{36}{72}}}{\sqrt{226.08}}\right) + 0.2\left(\frac{e^{-\frac{16}{72}}}{\sqrt{226.08}}\right)}$$

$$= \frac{0.8\,\frac{e^{-0.5}}{15.036}}{0.8\,\frac{e^{-0.5}}{15.036} + 0.2\,\frac{e^{-0.2}}{15.036}}$$

$$= \frac{0.8\left(\frac{0.6065}{15.036}\right)}{0.8\left(\frac{0.6065}{15.036}\right) + 0.2\left(\frac{0.8007}{15.036}\right)}$$

$$= \frac{0.8(0.0403)}{0.8(0.0403) + 0.2(0.0533)}$$

$$= \frac{0.0322}{0.0322 + 0.0107} = \frac{0.0322}{0.0429}$$

$$= 0.7506 \approx \underline{75.06\%}$$

The probability for that a company will issue a dividend this year given X=4 last year is at approx 75%.

ch 4 no. 9

a) Slide 24 reads: "odds = a measure of the likelihood of an outcome calculated as the ratio of the probability the outcome occurs over the propability it doesn't" $\Rightarrow \frac{Pr(Y=1|X) \rightarrow p(\text{outcome occurs})}{1-Pr(Y=1|X) \rightarrow p(\text{outcome doesn't occur})}$

The odds are at 37% → 0.37 which is equivalent to

$$0.37 = \frac{Pr(Y=\text{default}|x)}{1-Pr(Y=\text{default}|x)} \quad (\Rightarrow \text{same as: } Pr(Y=\text{not default}|x)$$

$$\Leftrightarrow \quad 0.37(1-Pr(Y=\text{default}|x)) = Pr(Y=\text{default}|x)$$

$$\Leftrightarrow \quad 0.37 - 0.37\,Pr(Y=\text{default}|x)) = Pr(Y=\text{default}|x)$$

$$\Leftrightarrow \quad 0.37 = 1.37\,Pr(Y=\text{default}|x)$$

$$\Leftrightarrow \quad Pr(Y=\text{default}|x) = \frac{0.37}{1.37} = 0.27007299$$

Only ≈ 27% of people with an odds of 37% of defaulting on their credit card will actually default.

b) Before it asked about the fraction of people, now it asks for the odds of an individual to default which is as the definition on top says literally just the probability that she defaults over the probability that she doesn't:

$$\frac{Pr(Y=\text{default}|X)}{Pr(Y=\text{not default}|X)} = \frac{0.16}{1-0.16} = \frac{0.16}{0.84} \approx 0.1905$$

Her odds of defaulting are at approx. 19 per cent.

```
# predict whether a given car gets high or low gas mileage

path_1 = '/Users/wasilengel/Desktop/School/Harris/Machine Learning/Auto-and-Default/Data-Auto.csv'

df = pd.read_csv(path_1)

df.head()

# a

df["mpg"].median() # median is at 22.75

#if df["mpg"] > 22.75:
#    df["mpg01"] == 1
#else:
#    df["mpg01"] == 0

df['mpg01'] = pd.Series(np.zeros(df.shape[0]))
df.loc[df['mpg']>22.75, 'mpg01'] = 1
df.loc[df['mpg']<=22.75, 'mpg01'] = 0
df.tail(10)

# Test
df['mpg01'].unique()
# It worked!

# b

# df
# Note: 11 columns in total

df.columns

for col in df.iloc[:,1:10].columns:
    sns.scatterplot(df[col],df['mpg01'])
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('mpg01')
    plt.show()

# Among the other variables in the dataset, most useful in predicting mpg01 are (in descending order):
```

```
# - horsepower: fairly good predictor where horsepower values above approx. 75 are
associated with mpg01 = 0.0 and horsepower values below approx. 140 with mpg01 = 1.0
# - weight: similarly good predictor like the pattern in horsepower where weight values above
approx. 2100 are associated with mpg01 = 0.0 and weight values below approx. 4000 with
mpg01 = 1.0
# - acceleration: again, similarly good predictor in that acceleration values below approx. 20.0
(with a couple exceptions) are associated with mpg01 = 0.0 and acceleration values above
approx. 11 with mpg01 = 1.0
# - displacement: only very few displacement values above approx. 200 seem to be associated
with mpg01 = 1.0
# The following variables are not useful since they do not show any pattern:
# - cylinders
# - year
# - origin
# - name
# mpg: obviously, there's a clear correlation because that's the base variable for mpg01 where
I can see the cut-off point is at 22.75 where everthing less is being coded as zero, and
everything more as one -- because of perfect multicollinearity, however, not useful!

# Overall, these findings make sense as mileage is associated with horsepower, weight,
acceleration capacities, and overall displacement rather than cylinders, or the car name/
origin.

# c

X = df.drop(['mpg01', 'mpg', 'cylinders', 'year', 'origin', 'name'], axis=1)
# dropping the ones I found were least associated with mpg01
Y = df['mpg01']

print(X.shape)
print(Y.shape)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=123)

# Y_train = Y_train.values.reshape(-1, 1)

# d

# Note that I already dropped the non- or least-associated variables with mpg01 in c

X_train.columns

print(X_train.shape)
print(Y_train.shape)
# note how training data has been reduced down to 80 per cent: from 392 to 313
```

```
# Choose method
lda_model = LinearDiscriminantAnalysis()

# Train model: fit X on Y
lda_model.fit(X_train, Y_train)

# Now, predict Y from test data
Y_pred = lda_model.predict(X_test)
Y_pred

# Calculate test error: that is, how much does Y_pred correctly identify Y_test?
score = accuracy_score(Y_test, Y_pred) # 0.8354430379746836
(1 - score) * 100
# That is equivalent to a test error of approx. 16.46 per cent.

# e

qda_model = QuadraticDiscriminantAnalysis()

qda_model.fit(X_train, Y_train)

Y_pred = qda_model.predict(X_test)
Y_pred

score = accuracy_score(Y_test, Y_pred) # 0.8607594936708861
(1 - score) * 100
# The test error is at approx. 13.92 per cent too.

# f

logit_model = LogisticRegression()

logit_model.fit(X_train, Y_train)

Y_pred = logit_model.predict(X_test)
Y_pred

score = accuracy_score(Y_test, Y_pred) # 0.8734177215189873
(1 - score) * 100
# Using logistic regression, the test error rate is at approx. 12.66 per cent.
```

<u>Ch 5</u> RESAMPLING METHODS

 # 5 **p.198f.**

```
path_2 = '/Users/wasilengel/Desktop/School/Harris/Machine Learning/Auto-and-Default/Data-Default.csv'

df = pd.read_csv(path_2)

df.head()

X = df.drop(['default', 'student'], axis=1)
X

Y = df['default']
Y.head(10)

d = {'Yes': True, 'No': False}

Y = Y.map(d)
Y.head(10)

# a

logit_all_model = LogisticRegression()

logit_all_model.fit(X, Y)

Y_pred = logit_model.predict(X)

score = accuracy_score(Y, Y_pred) # 0.9735

(1 - score) * 100
# Using logistic regression, the test error rate is at approx. 2.65 per cent.

# b (i)

print(X.shape)
print(Y.shape)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=123)

print(X_train.shape)
print(Y_train.shape)
```

```
# b (ii)

logit_model = LogisticRegression()

logit_model.fit(X_train, Y_train)

# b (iii)

y_posterior = logit_model.predict_proba(X_test)
y_posterior[:10]

# Convert to df
df_posterior = pd.DataFrame(y_posterior)
df_posterior.head(10)

# # Given that the columns represent the probability for label 0 and 1 respectively, I only care about the second column
df_posterior["defaults"] = df_posterior[1]>0.5
df_posterior.head(10)

# Make sure that there are some true values in there too:
df_posterior["defaults"].unique()

# The predicted default status is given by the new columns "defaults" in df_posterior and this vector here:
Y_pred = df_posterior["defaults"]
Y_pred.head(10)

# b (iv)

score = accuracy_score(Y_test, Y_pred) # 0.974
(1 - score) * 100
# The validation set error is at approx. 2.6 per cent.

# c

## Expanding test set size to 50 per cent of all observations

X_train1, X_test1, Y_train1, Y_test1 = train_test_split(X, Y, test_size=0.50, random_state=123)

print(X_train1.shape)
print(Y_train1.shape)

logit_model = LogisticRegression()
```

```
logit_model.fit(X_train1, Y_train1)

y_posterior = logit_model.predict_proba(X_test1)

df_posterior = pd.DataFrame(y_posterior)

df_posterior["defaults"] = df_posterior[1]>0.5

Y_pred = df_posterior["defaults"]

score = accuracy_score(Y_test1, Y_pred)
score

(1 - score) * 100

# The validation set error decreases for a test set size of 50 per cent to 2.5 per cent.
# Given the U-shape of the bias-variance trade-off, as the variance in our model increases,
# the bias, or test error rate, may first decrease (depending on how complex our model is to
# begin with). That illustrates how a higher variability is associated with more noise, which
# may later change because the validation estimate of the test error rate is a function of how
# we partition our data (see examples of that here below).

## Expanding test set size to 99 per cent of all observations

X_train3, X_test3, Y_train3, Y_test3 = train_test_split(X, Y, test_size=0.99, random_state=123)

print(X_train3.shape)
print(Y_train3.shape)

logit_model = LogisticRegression()

logit_model.fit(X_train3, Y_train3)

y_posterior = logit_model.predict_proba(X_test3)

df_posterior = pd.DataFrame(y_posterior)

df_posterior["defaults"] = df_posterior[1]>0.5

Y_pred = df_posterior["defaults"]

score = accuracy_score(Y_test3, Y_pred)
score
```

```
(1 - score) * 100

# The validation set error increases for a test set size of 99 per cent to 3.32 per cent.
# Given the U-shape of the bias-variance trade-off, as the variance in our model increases,
# the bias, or test error rate, may increase after it first decreases (depending on how
# complex our model is to begin with). Because of the high variance, there is too much noise
# now -- this comes at the detriment of the validation set error, that is, the bias goes up.

## Expanding test set size to 2 per cent of all observations

X_train2, X_test2, Y_train2, Y_test2 = train_test_split(X, Y, test_size=0.02, random_state=123)

print(X_train2.shape)
print(Y_train2.shape)

logit_model = LogisticRegression()

logit_model.fit(X_train2, Y_train2)

y_posterior = logit_model.predict_proba(X_test2)

df_posterior = pd.DataFrame(y_posterior)

df_posterior["defaults"] = df_posterior[1]>0.5

Y_pred = df_posterior["defaults"]

score = accuracy_score(Y_test2, Y_pred)
score

(1 - score) * 100

# The validation set error increases for a test set size of 2 per cent to 4.5 per cent.
# Given the U-shape of the bias-variance trade-off, we are now on the far left side so the
# variance in our model is low and as such, the prediction error of our validation set is
# high (danger of overfitting). From there, the bias then decreases with increased variance
# (see test set size of 50 per cent) before it climbs again (see test set size of 99 per cent).

# d

## Prepare data

df.head(10)

X = df.drop(['default'], axis=1)
```

```
X.head(10)

e = {'Yes': True, 'No': False}

X["student"] = X["student"].map(e)
X.head(10)

Y = df['default']
Y.head(10)

Y = Y.map(e)
Y.head(10)

print(X.shape)
print(Y.shape)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=123)

print(X_train.shape)
print(Y_train.shape)

## Perform analysis

logit_model = LogisticRegression()

logit_model.fit(X_train, Y_train)

y_posterior = logit_model.predict_proba(X_test)

df_posterior = pd.DataFrame(y_posterior)

df_posterior["defaults"] = df_posterior[1]>0.5

Y_pred = df_posterior["defaults"]

score = accuracy_score(Y_test, Y_pred)
score

(1 - score) * 100
# The validation set error is at approx. 3 per cent now.

# So, compared to b), adding an independent variable for being a student leads to an slight
# increase in the validation set error from 2.6 per cent in b) to approx. 2.75 per cent here.
# However, it doesn't seem that adding the student dummy changes the results significantly.
```