**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**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**# 7 p.170**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**# 9 p.170**

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**# 11 p.171f.**

# 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.

**Ch 5 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.