Lab 1 - Machine Learning for Social Science (Solutions)

To be handed in no later than August 27th, 13:00. The submission should include code, relevant output, as well as answers to questions. We recommend the use of RMarkdown to create the report.

Part 1: Bernie Sanders and Donald Trump tweets.

For the first part of the lab, you will work with a dataset containing a sample of tweets from Donald Trump and Bernie Sanders. The objective is to explore how accurately we can predict who the author of a given tweet is based on its content, and to identify which words are the most discriminative.

The tweets have been preprocessed & cleaned for you, and are stored in a so-called document-term matrix format with rows indicating tweets, and columns indicating the frequency of words in different tweets.

1. Begin by importing the file "`trumpbernie.csv`" (hint: for example by using `data.table`'s `fread()` function or the standard `read.csv()`). Report how many *rows* and *columns* there are in this dataset (hint: you may for example use `dim()`). Would you characterize this data set as being *high-dimensional* or *low-dimensional*? Based on this, do you expect that a standard logistic regression will work well for the purpose of prediction?

```
# Load R packages
library(data.table)

# Import data
trumpbernie <- fread(input = '/Users/marar08/Documents/Teaching/MLSS_HT2025/Labs/W1/final/2025/trumpberr

# Examine dimensionality of data
dim(trumpbernie)
```

```
## [1] 1003 1496
```

As we discussed in the lecture, data sets which a large number of variables relative to the number of rows are considered to be high-dimensional. In this case, as we have more variables than rows, this is certainly high-dimensional. With regards to the appropriateness of logistic regression in this setting: From the lecture, we know that standard linear models a) cannot estimate more than n parameters, and b) that when $p \geq n$, such models will perfectly fit the training data—and thus produce considerable overfitting. Hence, this information leads me to believe that a standard logistic regression model will not produce good predictions here.

2. Estimate a *standard logistic regression model* on the whole data set using the `glm` function (recall to specify `family="binomial"` in `glm`). The outcome variable is `trump_tweet`, and the rest of the columns (the word frequencies) are the input variables. Note: estimating the model may take a couple of minutes. When the estimation of the model is finished, do the following:

a. Extract the coefficients from the estimated model using the `coef()` function and inspect the coefficients that are placed 1010–1050 in the output from `coef()` (hint: to inspect the coefficients placed 1010–1050 you may use standard brackets, e.g., `coef(mymodel)[1010:1050]`). Do you notice anything special?
b. Examine the *training* accuracy of the estimated model. What does this result suggest about the predictive capacity of the model? You may use the following code to compute the training accuracy:

```
# 2) Estimate standard logistic regression
myglm <- glm(data = trumpbernie,
             trump_tweet ~ .,
             family = 'binomial')
```

```
## Warning: glm.fit: algorithm did not converge
```

```
# a) Extract specified coefficients
print(coef(myglm)[1010:1050])
```

```
##     prefer    premium     prepar       pres  prescript    present     presid
##         NA         NA         NA         NA         NA         NA         NA
## presidenti      press    pressur     pretti    prevent   previous      price
##         NA         NA         NA         NA         NA         NA         NA
##    primari      prime   prioriti     prison     privat   privileg        pro
##         NA         NA         NA         NA         NA         NA         NA
##    probabl    problem    process   proclaim     produc    product     profit
##         NA         NA         NA         NA         NA         NA         NA
##    program   progress    project     promis     proper     propos    protect
##         NA         NA         NA         NA         NA         NA         NA
##    protest      proud     provid     public     puerto       pull
##         NA         NA         NA         NA         NA         NA
```

All the coefficients numbered 1010–1050 are "'NA"'. This, as remarked upon in #1 is expected as standard linear models can only estimate n parameters. As the variables are ordered by alphabetical order, this becomes a very arbitrary type of variable selection. Not good.

```
# b) Examine accuracy on training data
comparison_df <- data.frame(train_predictions=myglm$fitted.values,
                            observed=myglm$y)
# Apply prediction threshold
comparison_df$train_predictions<-ifelse(comparison_df$train_predictions>=0.5,
                                        yes = 1,
                                        no = 0)
# Compute accuracy (scale: 0-1, 0=0%, 1=100%)
nrow(comparison_df[comparison_df$train_predictions==comparison_df$observed,]) /
  nrow(comparison_df)
```

```
## [1] 1
```

The accuracy on the training set is 100%. Without any other information, this is suspiciously high—and one might reasonably suspect that we are *overfitting*. Knowing that we have fitted a standard linear model to a data set with more columns than rows, we know that we have overfitted the data. In other words, this suggests that predictions from this model are not likely to be good on test data.

3. Use the `caret` package to implement a 3-fold cross-validation procedure that estimates the *test* accuracy of a *standard logistic regression* (hint 1: two functions are relevant here: `trainControl` and `train` | hint 2: specify `method="glm"` and `family='binomial'` in `train` to fit a standard logistic regression). Note, before you run `train()`, make sure you have formatted the outcome variable `trump_tweet` as a `factor` variable (this is needed for the "caret"'package to recognize the problem as a classification problem). Report the accuracy. Does this result align with your expectations from #1 and #2? Do the results from #2 and #3 provide any indications of either over- or underfitting?

```
# Load caret package
library(caret)

# Set resamplig settings
tc <- trainControl(method = 'cv', number = 3)

# Format outcome variable as factor
trumpbernie[,trump_tweet := as.factor(trump_tweet)]
```

```
# Run cross-validation-glm-estimation procedure
tc_glm <- train(trump_tweet ~ .,
                method='glm',
                data = trumpbernie,
                family='binomial',
                trControl=tc)
```

```
# Extract results
tc_glm$results
```

```
##   parameter  Accuracy     Kappa AccuracySD    KappaSD
## 1      none 0.5005273 0.0014389 0.03901526 0.07773997
```

The test accuracy is ≈ 50%, which is a massive reduction compared to 100%. Indeed, we would do just as well predicting by flipping a coin. In other words, this confirms the expectations from 1–2: our model is considerably overfitted.

4. Now we shall move beyond the standard logistic regression, and more specifically, turn to ridge regression for our prediction task. This importantly entails deciding on a value for the parameter λ. Use `glmnet`'s function `cv.glmnet` to find the λ that minimizes the test error, and report the associated test accuracy. Is this a better or worse prediction model compared to the one in #2–3? Which of the two models do you believe have a higher variance? Why?

    - When specifying `cv.glmnet`'s arguments, note the following: (i) Unlike `glm` and `train`, the `cv.glmnet` function does not have a `data` argument. Instead, you have to specify `x` and `y` separately (hint: you can for example use `$` to extract and delete columns). (ii) `alpha` dictates the model type (0=ridge, 1=lasso), and `standardize` whether you want to standardize the data prior to estimation (which we *do*). (iii) Finally, set the number of folds to 5 (`nfolds=5`), `family`="binomial" (to indicate that `y` should be treated as a binary variable), and `type.measure="class"` (to retrieve a measure of *accuracy*).

```
# Load R package
library(glmnet)

# Extract y
y <- trumpbernie$trump_tweet
```

```r
# Extract X and make into matrix
X <- trumpbernie[,-c('trump_tweet'),with=F]
X <- as.matrix(X)
```

```r
# Perform CV to identify best lambda
set.seed(1234)
myglmnet <- cv.glmnet(x = X,
                      y = y,
                      alpha = 0,
                      standardize = TRUE,
                      nfolds = 5,
                      type.measure="class",
                      family='binomial')
```

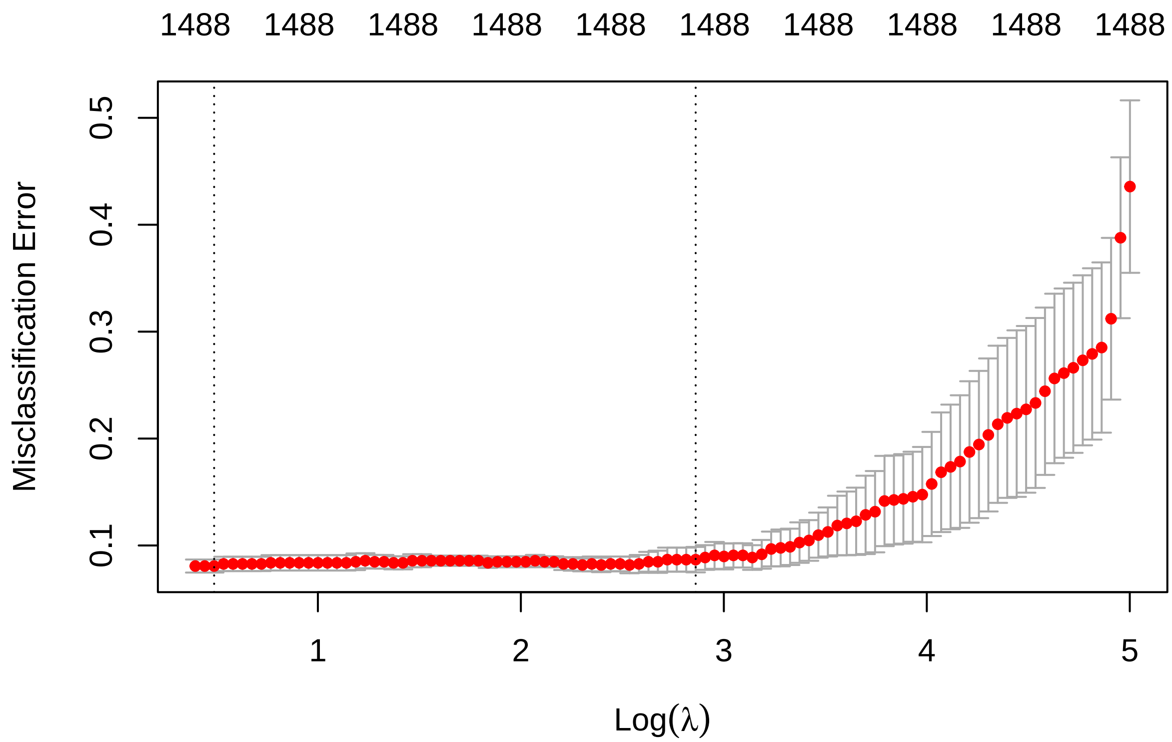
```r
# Extract classification error
print(myglmnet)
```

```
##
## Call:  cv.glmnet(x = X, y = y, type.measure = "class", nfolds = 5, alpha = 0,      standardize = TRUE
##
## Measure: Misclassification Error
##
##     Lambda Index Measure       SE Nonzero
## min   1.63    98 0.08076 0.006102    1488
## 1se  17.48    47 0.08674 0.011883    1488
```

This is indeed a better model than the one estimated in 1–2. This is because what we care about is the test error/accuracy: ridge ≈ 92% test accuracy compared to ≈ 50% for the standard logistic regression. The latter model is the one with the highest variance. The difference in accuracy on training data and test data for the standard logistic regression model is dramatic (100% vs. 50%).

5. Plot lambda against the classification error (hint: just `plot()` the object which you stored the output from `cv.glmnet`). Interpret the plot in terms of bias and variance.

```r
# Plot classification error against lambda
plot(myglmnet, sign.lambda = 1)
```



The misclassification error is minimized when $\lambda \approx 1.7$. This is the λ which balances the trade-off between bias and variance the best. If we increase the value of λ more (going to the right in this plot), the bias is increased more than the variance is reduced, and hence our overall test error is degraded. Conversely, if we reduce λ (going left in the plot), the variance increases more than the bias is reduced. Although note that (a) the range considered to the left is very small, and (b) the change in error is minimal.

6. Lastly, extract the coefficients associated with the lowest test error (hint: you can use `coef(myfit, s='lambda.min'` for this). Have a closer look at the coefficients with the largest *positive* and largest *negative* values. What do they reveal? Do the words you find on either side confine to your expectations?

```
# Extract coefficients associated with lowest test error
mycoefs <- coef(myglmnet, s = 'lambda.1se')
mycoefs_dt <- data.table(var=rownames(mycoefs),
                         coef=mycoefs[,1])
X_sd <- apply(X, 2, sd)
X_sd <- data.table(var=names(X_sd),sd=X_sd)
mycoefs_dt <- merge(x=mycoefs_dt,
                    y=X_sd, by = 'var')
mycoefs_dt[,coef_std := coef*sd]
mycoefs_dt <- mycoefs_dt[order(coef_std,decreasing = T)]
# Print most positive (std) coefs
print(head(mycoefs_dt,15))
```

```
##          var        coef         sd     coef_std
##       <char>       <num>      <num>        <num>
```

```
## 1:    great 0.021005391 0.3763072 0.007904480
## 2:     news 0.022109245 0.2544482 0.005625658
## 3:     fake 0.024616738 0.2227188 0.005482611
## 4: democrat 0.016195433 0.3307572 0.005356756
## 5:   border 0.019924968 0.2535315 0.005051607
## 6:      dem 0.025235349 0.1788237 0.004512679
## 7:     hunt 0.025810618 0.1589826 0.004103439
## 8:    witch 0.025736305 0.1589826 0.004091625
## 9:    media 0.020592636 0.1896574 0.003905546
## 10:   china 0.015731323 0.2473896 0.003891766
## 11:    good 0.014747986 0.2620121 0.003864151
## 12:  report 0.019201693 0.1951547 0.003747300
## 13:    mani 0.014662518 0.2547804 0.003735722
## 14:    will 0.006754833 0.5495436 0.003712075
## 15:    just 0.012765243 0.2906152 0.003709774
```

```
# Print most negative (std) coefs
print(tail(mycoefs_dt,15))
```

```
##             var        coef        sd     coef_std
##          <char>       <num>     <num>        <num>
## 1:         wage -0.01986115 0.1945214 -0.003863419
## 2:       climat -0.01988073 0.1951547 -0.003879817
## 3:       defeat -0.02226801 0.1817598 -0.004047429
## 4:       afford -0.02350831 0.1761835 -0.004141775
## 5:        stand -0.01706674 0.2437429 -0.004159897
## 6:       corpor -0.02274455 0.1921074 -0.004369397
## 7:       togeth -0.01846844 0.2377667 -0.004391180
## 8:   billionair -0.02533141 0.1734973 -0.004394931
## 9:      million -0.01506608 0.2981793 -0.004492392
## 10:        join -0.02419452 0.1861145 -0.004502950
## 11:      worker -0.01445558 0.3117618 -0.004506698
## 12:        live -0.01922487 0.2544482 -0.004891733
## 13:    movement -0.02461862 0.2034717 -0.005009193
## 14:        must -0.01502762 0.3425647 -0.005147933
## 15:      health -0.01883941 0.2828559 -0.005328838
```

```
# Print most positive (non-std) coefs
mycoefs_dt <- mycoefs_dt[order(coef,decreasing = T)]
print(head(mycoefs_dt,15))
```

```
##             var       coef         sd     coef_std
##          <char>      <num>      <num>        <num>
## 1:      atlanta 0.02862931 0.03157545 0.0009039834
## 2:         Npme 0.02818137 0.04463214 0.0012577949
## 3:      patriot 0.02795459 0.03157545 0.0008826788
## 4:     colorado 0.02789429 0.03157545 0.0008807747
## 5:     sacrific 0.02764327 0.05463567 0.0015103086
## 6:       confid 0.02759155 0.05463567 0.0015074827
## 7:        shape 0.02751240 0.05463567 0.0015031584
## 8:      liberti 0.02740018 0.05463567 0.0014970276
## 9:       prayer 0.02735563 0.07046378 0.0019275811
## 10:      plenti 0.02730650 0.05463567 0.0014919088
```

```
## 11:       extend 0.02724666 0.05463567 0.0014886396
## 12:       talent 0.02719140 0.05463567 0.0014856205
## 13: humanitarian 0.02717716 0.07046378 0.0019150058
## 14:       spirit 0.02716106 0.06305629 0.0017126754
## 15:       regard 0.02714690 0.05463567 0.0014831894
```

```r
# Print most negative (non-std) coefs
print(tail(mycoefs_dt,15))
```

```
##          var        coef         sd      coef_std
##       <char>       <num>      <num>         <num>
##  1:    engag -0.02740433 0.04463214 -0.0012231138
##  2: xenophob -0.02744365 0.03157545 -0.0008665455
##  3:   caucus -0.02748394 0.04463214 -0.0012266672
##  4: overturn -0.02749701 0.06305629 -0.0017338594
##  5:       dr -0.02754035 0.05463567 -0.0015046858
##  6: franklin -0.02754570 0.04463214 -0.0012294236
##  7:    forum -0.02760551 0.05463567 -0.0015082457
##  8:  largest -0.02760653 0.05463567 -0.0015083012
##  9:     born -0.02761103 0.04463214 -0.0012323392
## 10:     youv -0.02763923 0.04463214 -0.0012335979
## 11:    petit -0.02767433 0.05463567 -0.0015120056
## 12:   vulner -0.02790682 0.05463567 -0.0015247079
## 13:    visit -0.02795125 0.04463214 -0.0012475243
## 14:     view -0.02798920 0.04463214 -0.0012492177
## 15:   volunt -0.02805614 0.03157545 -0.0008858853
```

Conclusion based on non-standardized coefficients (old): Although we can find some words which we reasonably can expect to be used more by conservatives and appear on Trump's side ("patriot","sacrifice","liberty","prayer","talent") and some which we expect to be more used by democrats and Bernie ("volunteer","vulnerable","dr","xenophobia"), the pattern is not super clear. I suspect that this has to do with the fact that we have a rather small randon sample, and that thus, event-driven differences (having a rally in a partiular city) can show up as the more important.

Conclusion based on standardized coefficients (updated): The results indeed align with my expectations. The words most predictive of a Trump tweet include "fake","news","witch","hunt". For Bernie, we instead find words like "afford","billionare","corporate","climate".

Part 2: Social Network Ad Purchase

For the second part of the lab, you will work with a dataset that contains information about individuals' purchasing behavior under exposure to online ads. The data originates from an online shopping site, and can be downloaded from *Kaggle*. Our goal is to examine how well we can predict purchases on the basis of `Age`, `Gender` and `Salary`, and to explore the character of the associations between these variables and the outcome.

1. Begin by importing the file "`Kaggle_Social_Network_Ads.csv`". Format the outcome variable `Purchased` as a `factor` variable (this is required for the subsequent analysis using the `caret` package).

```r
# Import data
snaads <- fread(input = '/Users/marar08/Documents/Teaching/MLSS_HT2025/Labs/W1/final/2025/Kaggle_Social_

# Format outcome variable as factor
snaads[,Purchased := as.factor(Purchased)]
```

2. Use the `caret` package to implement a 5-fold cross-validation that assesses the test accuracy of a *standard logistic regression model* (hint: two functions are relevant here: `trainControl` and `train`). Report its test accuracy. Note: To ensure that the folds generated by `caret` are identical for different models, add a `set.seed(12345)` above each use of `train()`.

```
# Set resampling options (5-CV)
tc <- trainControl(method = 'repeatedcv', number = 5, repeats = 50)

# Assess test accuracy of standard logit using CV
set.seed(12345)
base_model <- train(Purchased ~ Age + Salary + Gender,
               data = snaads,
               method = "glm",
               family='binomial',
               trControl = tc)

# Extract results
base_model$results
```

```
##   parameter  Accuracy     Kappa AccuracySD    KappaSD
## 1      none 0.8463568 0.6545754 0.03750781 0.08515359
```

The standard logistic regression obtains a test accuracy of $\approx 85\%$.

3. To investigate whether *GAMs* can improve the performance over the standard logistic regression, implement three separate 5-fold cross-validations; each estimating a GAM with a different degree of freedom for the natural cubic splines ($df \in \{2, 3, 4\}$). Create splines for the two variables *Age* and *Salary*, but not for *Gender*, which is a categorical variable (hint: use the `ns()` function from the `splines` package to create splines). Again, to ensure identical folds, add `set.seed(12345)` above each `train()`. You may also re-use the `trainControl` object from the previous task. Report the accuracies of the different models. Having do so, answer the following questions:

a. Do you observe any improvement compared to the standard logistic regression?

Yes. The test accuracy increases to $\approx 90\%$.

b. What does the difference in performance between the standard logistic regression and the GAMs suggest about the former? Is it over- or underfitted? Does it have high(er) bias or high(er) variance compared to the GAMs?

It suggests that the standard linear regression is underfitting the data. It is not capturing the patterns of the data sufficiently well; and thus has a higher bias compared to the GAMs.

c. Which of the three GAMs do you prefer? Motivate.

Because the difference in accuracy is rather marginal, I would pick the GAM with 2 degrees of freedom. The reasoning for this is that, if two models perform similarly, but one is "simpler" than the other; then the similer model runs a lower risk of overfitting. In a real-world scenario, we might consider computing the standard errors around the estimated mean test accuracies—enabling statistical testing of significant differences.

```r
# Load package
library(splines)

# Fit GAM with 2 degree natural cubic splines
set.seed(12345)
gam2 <- train(Purchased ~ ns(Age,2)+ns(Salary,2)+Gender,
              method='glm',
              family='binomial',
              data=snaads,
              trControl = tc)
# Fit GAM with 3 degree natural cubic splines
set.seed(12345)
gam3 <- train(Purchased ~ ns(Age,3)+ns(Salary,3)+Gender,
              method='glm',
              family='binomial',
              data=snaads,
              trControl = tc)
# Fit GAM with 4 degree natural cubic splines
set.seed(12345)
gam4 <- train(Purchased ~ ns(Age,4)+ns(Salary,4)+Gender,
              method='glm',
              family='binomial',
              data=snaads,
              trControl = tc)
gam2$results
```

```
##   parameter  Accuracy     Kappa AccuracySD    KappaSD
## 1      none 0.8968595 0.7728214 0.02932001 0.06471985
```

```r
gam3$results
```

```
##   parameter  Accuracy    Kappa AccuracySD    KappaSD
## 1      none 0.8964145 0.772301 0.03065777 0.06745364
```

```r
gam4$results
```

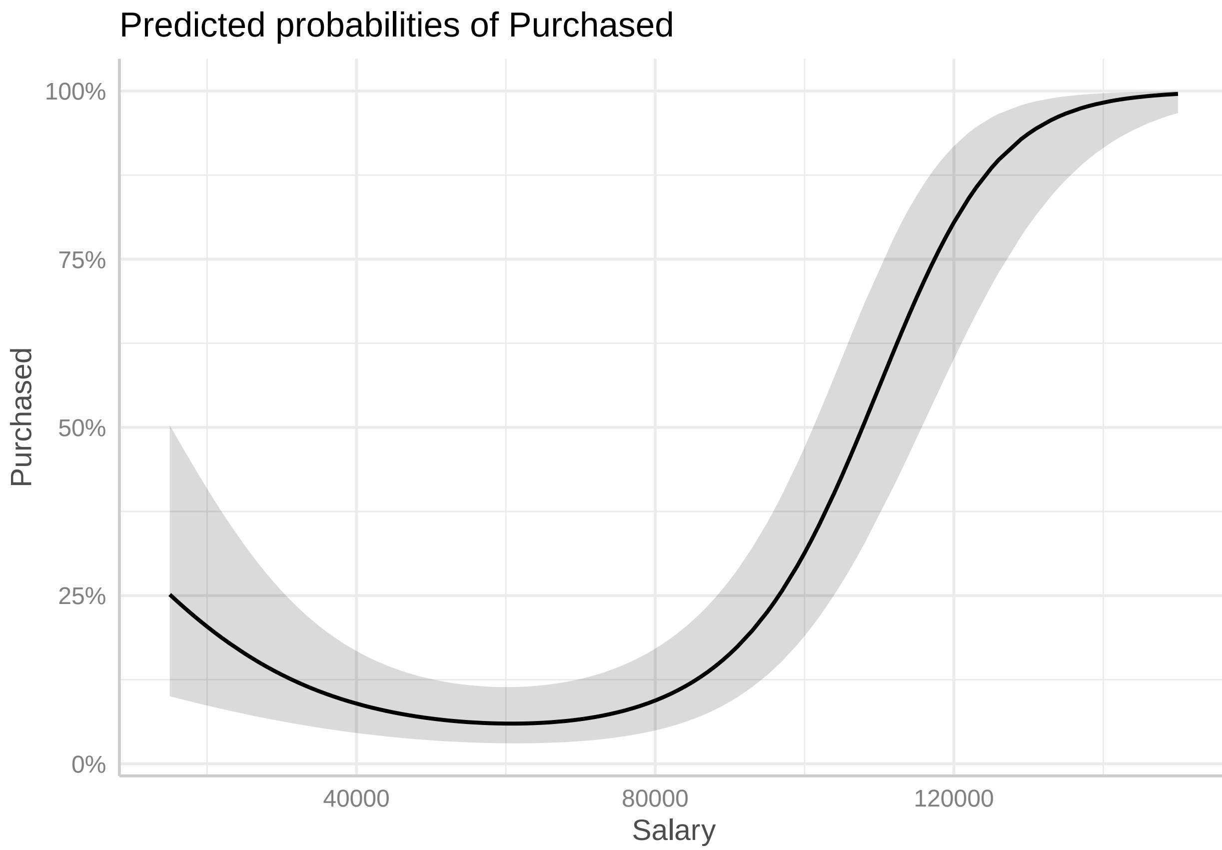
```
##   parameter Accuracy     Kappa AccuracySD    KappaSD
## 1      none 0.903623 0.7907388 0.02872942 0.06219691
```

4. Next, you shall examine the predictive relationships between the two continuous variables (*Age* and *Salary*) and the outcome. For this, you will use `ggeffects`'s `ggpredict()` function which computes predictions while *varying one* variable and holding the *remaining fixed* at their means/mode. To do so, first re-estimate the GAM-specification that you found to be the best, on the full data using `lm` (`ggeffects` does not accept objects from `caret`). Interpret. Do you find any non-linear relationship?

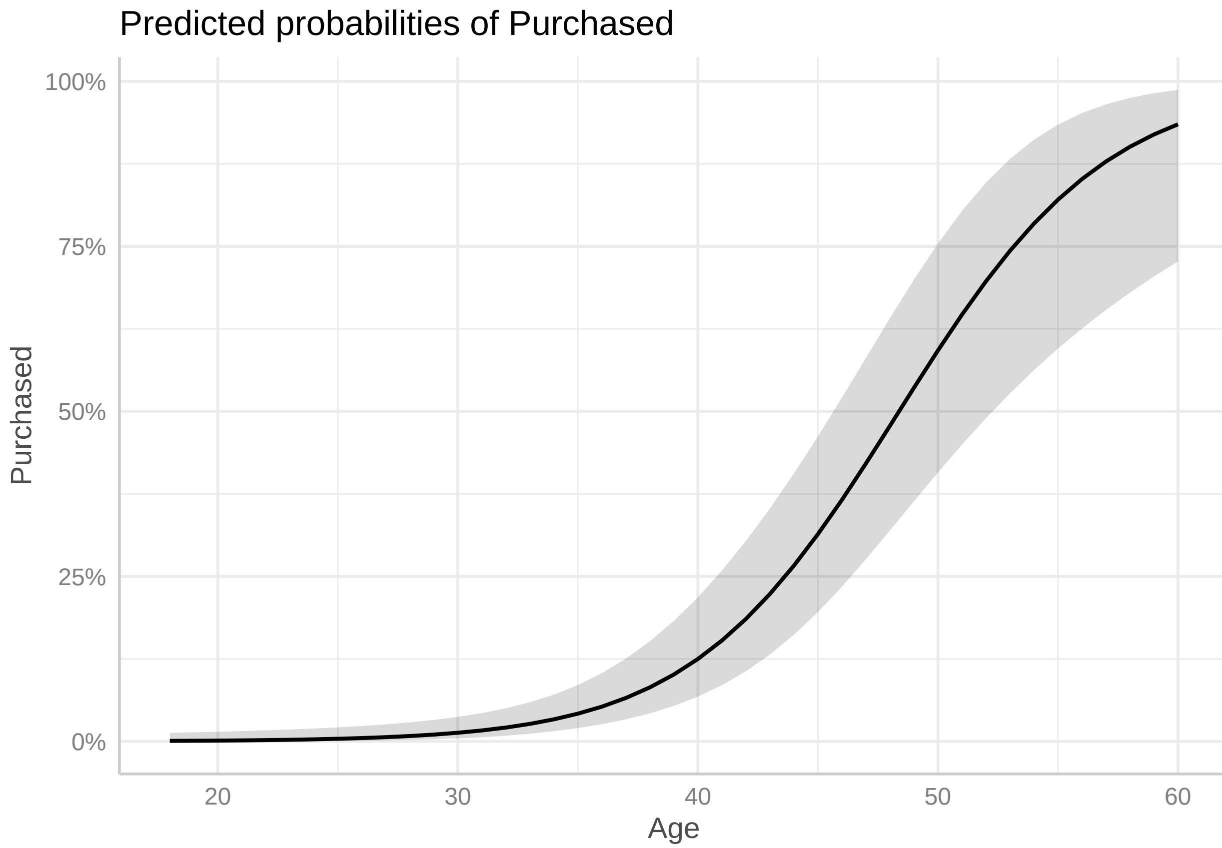
```r
# Load package
library(ggeffects)

# Re-estimate best specification on full data
gam2 <- glm(formula = Purchased ~ ns(Age,2)+ns(Salary,2)+Gender,
            data = snaads,
```

```r
           family = 'binomial')
# Plot marginal predictions from model
plot(ggpredict(model = gam2,terms = 'Salary'))
```




```r
# Plot marginal predictions from model
plot(ggpredict(model = gam2,terms = 'Age'))
```



The relationship does indeed appear non-linear for both "'Age"' and "'Salary"'. Both exhibit either an approximately constant ("'Age"') or negative ("'Salary"') relationship for the first part of the x-axis, but then suddenly exhibit sharp increases.

5. In this second part of the lab, we used GAMs to improve predictive performance. Would we expect to see similar improvements if we instead had used *ridge* and *lasso* regression? Why/why not?'

No, *ride* or *lasso* would not, by themselves improve performance here. This is because what our baseline standard linear model suffers from in this case is underfitting (high bias). Ridge/lasso help address situations where our models have high variance (overfitting).[1]

[1] A caveat here is that ridge/lasso—or reguarizastion more generally—could potentially be used in combination with the creation of a large nuber of non-linear basis functions and interactions.