NOTES: The standard approach in estimating the probability of selecting a particular word has two primary drawbacks: It omits cases where a word is included in the test but not the training set, and invokes bias in the case of small sample sizes. smooth_count is an added feature that enables us to provide a smoothing factor to marginally reduce the effect of new words added to the model as its size increases. This method allows us to add a probability to estimating observations regardless of the distribution of already existing observations.

```
{r}
smooth_count = 1/nrow(X)
w_AP = colSums(AP_train + smooth_count)
w_AP = w_AP/sum(w_AP)
```

- Laplasian smoothing: r = 1/N as a smoothing factor, where N - number of documents in the collection. More documents = smaller smoothing factor

Helpful Tools:

a.) Readerplain specifies certain flags, e.g English language.

```
{r}
# Need a more clever regex to get better names here
all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))
```

b.) Before reading in a corpus (list of texts from R), first initiate the text mining function in library(). Proceed by using `Corpus(VectorSource())`to turn the corpus into a `tm` `Corpus` object.

```
{r}
library(tm)
my_corpus = Corpus(VectorSource(all_docs))
names(my_corpus) = file_list
```

c.) Continue by running through several transformers to filter out clutter and make the data more analysis-friendly. Components to be removed include capitalization, numbers, punctuation, white space, and overused insignificant words.

```
{r}
# Preprocessing
my_corpus = tm_map(my_corpus, content_transformer(tolower)) # make everyt
hing lowercase
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers)) # remov
e numbers
my_corpus = tm_map(my_corpus, content_transformer(removePunctuation)) # r
emove punctuation
my_corpus = tm_map(my_corpus, content_transformer(stripWhitespace)) ## re
move excess white-space
my_corpus = tm_map(my_corpus, content_transformer(removeWords),stopwords
("SMART"))
```

d.) stopwords = function to scrub out verbal filler, distractions. Bottom line: Remove barebones typical words to filter out the unique elements

```
{r}
stopwords("en")[1:10]
my_documents = tm_map(my_documents, content_transformer(removeWords), sto
pwords("en"))
```

e.) Convert Corpus to DocumentTermMatrix (DTM) to make it easier to run numerical operations in the model. We can use DTM to remove sparse values from the matrix.

```
{r}
DTM = DocumentTermMatrix(my_corpus)
DTM # some basic summary statistics

class(DTM)  # a special kind of sparse matrix format

## You can inspect its entries...
inspect(DTM[1:10,1:20])

DTM_simon = removeSparseTerms(DTM_simon, 0.95)
```

f.) X = as.matrix(DTM) to form a dense matrix

```
{r}
X = as.matrix(DTM)
```

g.) Use naïve bayes model to calculate and compare log probabilities of each vector.

```
{r}
# AC's multinomial probability vector
w_AC = colSums(AC_train + smooth_count)
w_AC = w_AC/sum(w_AC)
```

Principal Component Regression:

Technique enables analysis of large sets of strongly correlated features.

PCR works by extracting scores from principal components, and convert the values into PC scores.

```
{r}
gasoline = read.csv('../data/gasoline.csv', header=TRUE)
head(gasoline)

X = as.matrix(gasoline[,-1])
y = gasoline[,1]


# Ordinary least squares
lm1 = lm(y ~ X)

# Yikes!
summary(lm1)
dim(X)
```

Extract the first K (in this case 5) Principal Components before computing the scores of all the component observations. These scores in turn become the new features to fit the model. This method can be applied without affecting the fit nor the R-squared value of the model.

```
{r}
# Regress on the first K
K = 5
V = pc_gasoline$rotation[,1:K]
scores = X %*% V
pcr1 = lm(y ~ scores)

summary(pcr1)

plot(fitted(pcr1), y)

# Express the coefficients in terms of the original variables
beta_pcr = coef(pcr1)[-1]
beta_nir = rowSums(V %*% diag(beta_pcr))

plot(beta_nir)

# This coefficient vector implies the same fitted values
plot(X %*% beta_nir, scores %*% beta_pcr)
abline(0,1)
```

- Take-home message: Principal component analysis enables reduced feature set.