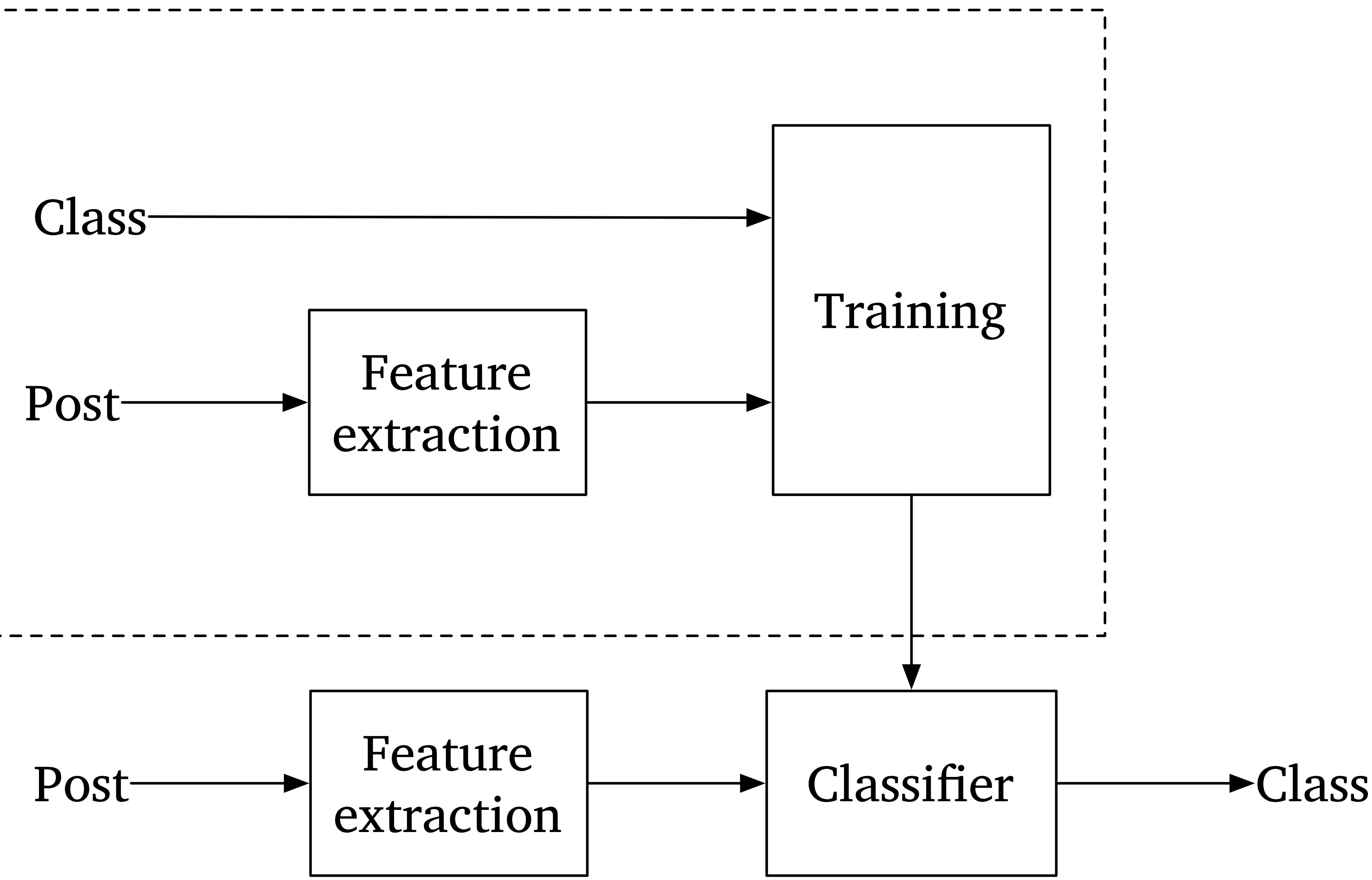
# Homework

- Read chapters 1, 2, 5 6, 7 in *Introduction to Machine Learning with Python*

# Text classification



# Bayesian models

- Suppose we have a representation of a text as word vector $x$ and we want to predict its class $c$
- If we have a way of modeling $P(c|x)$, the **Bayes Decision Rule** says our prediction should be:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(c|x)$$

- **Discriminative** methods directly model $P(c|x)$
- **Generative** methods model the joint probability $P(x, c)$

# Baseline classifier

- We get a 'baseline' for a classification task by simply assigning the most frequent class to each instance:

$$\hat{c} = \operatorname*{argmax}_{c \in C} P(c)$$

- Here we assume that $P(c|x) = P(c)$, i.e., $X$ and $C$ are independent
- Whatever information we can extract from $x$ should improve classification accuracy

# Naive Bayes classifiers

- A generative classifier requires a model of $P(x, c)$
- We can break this down into two parts: the **class prior** $P(c)$, and a **likelihood** $P(x|c)$

$$P(x, c) = \frac{P(c)\, P(x|c)}{P(x)}$$

- Since argmax $P(x, c)$ doesn't depend on $P(x)$, we'll ignore it
- The class priors $P(c)$ are easy to estimate from training data:

$$\hat{P}(c) = \frac{\text{\# of texts in class } c}{\text{\# of texts in total}}$$

# Naive Bayes classifiers

- To model $P(x|c)$, we can make the simplifying assumption that each of the features $x_i$ in $x$ are independent, so that:

$$P(x|c) = \prod_i P(x_i|c)$$

- Now we need to get estimates of $P(x_i|c)$
- Two event models: Bernoulli and multinomial

# Event models

- If we represent a document as a **set** of words, then each word $w_i$ corresponds to $x_i$ is a Bernoulli variable where $x_i$=1 if the document contains word $i$ and $x_i$=0 if it doesn't, then:

$$P(x_i|c) = P(x_i = 1|c)^{x_i} (1 - P(x_i = 1|c))^{1-x_i}$$

- And, we get $P(x_i{=}1|c)$ from the training data thusly:

$$\hat{P}(x_i = 1|c) = \frac{\text{\# of texts in class } c \text{ containing } x_i}{\text{\# of texts in } c}$$

- Note that this doesn't take into account the number of times a word appears in a text or the length of a text

# Event models

- If instead we represent a document as a **bag** of words, then we can model a document as a sequence of random draws from a multinomial distribution
- The probability of picking word *w* if the document class is *c* once is $P(w|c)$
- The probability of picking the word *x* times in a row is $\mathrm{P}(w|c)^x$
- The probability of drawing a collection of words *in that order* is:

$$\prod_i P(w_i|c_i)^{x_i}$$

# Event models

- This underestimates $P(w|c)$, since lots of ordered sequences correspond to the same bag of words
- How many different ways are there to draw word $w_1$ $x_1$ times, word $w_2$ $x_2$ times, and so on?
- We can use the **multinomial coefficient**

$$P(x|c) = \binom{\sum_i x_i}{x_1, x_2, \ldots} \prod_i P(w_i|c)^{x_i}$$

$$= \left(\sum_i x_i\right)! \prod_i \frac{P(w_i|c)^{x_i}}{x_i!}$$

# Event models

- The parameters of the multinomial model are the individual word probabilities $P(w_i|c_j)$
- We can estimate those from training data as:

$$\hat{P}(w_i|c) = \frac{\text{\# of times word } i \text{ occurs texts in class } c}{\text{\# of words in texts in } c}$$

# Text classification

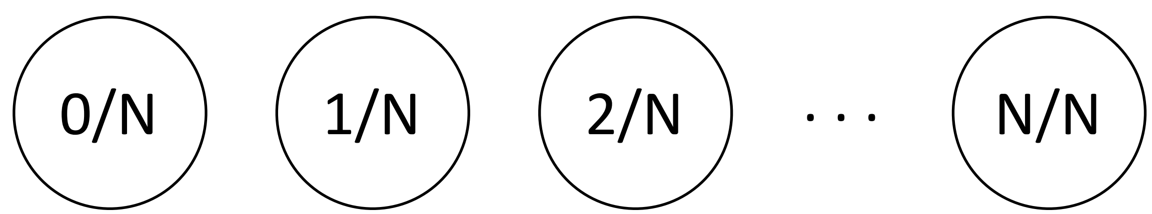
- The multinomial model takes word frequencies and document length into account, but treats multiple occurrences of a word as independent events
- McCallum and Nigam (1998) compare the two event models
- Multinominal model almost always outperforms multivariate Bernoulli model, by 25% or so
- The multinominal model handles large vocabulary sizes much better
- It’s easier to see how to add non-text features and to account for limited inter-dependencies using a multivariate Bernoulli model

# Parameter estimation

- Event models give us a general family of distributions
- The specific distribution depends on the parameters $P(c)$ and $P(w_i|c)$
- The **maximum likelihood estimate** (MLE) finds parameter values that predict that the training corpus we have is the most likely training corpus
- A big problem: MLE predicts that anything that didn't happen in the training data can never happen
- Can we do better?

# Laplace's Law of Succession

- Laplace (c. 1775) was interested in inductive reasoning: what can we conclude from a sequence of observations?
- Suppose we've got a set of coins labeled $0, \ldots, N$ such that coin $i$ comes up heads with probability $i/N$




- Given that you've seen one of these coins come up heads $n$ times in a row, which coin are you looking at?

# Laplace's Law of Succession

- The probability of getting heads on the next toss is:

$$P(n+1|n) = \frac{P(n+1)}{P(n)}$$

- If we take the limit as $N \to \infty$, then

$$\lim_{N\to\infty} P(n+1|n) = \frac{n+1}{n+2}$$

- If we've seen 10 heads, the probability that the next toss will be heads is (cf. MLE)

$$\frac{10+1}{10+2} = \frac{11}{12}$$

# Laplace's Law of Succession

- Laplace's Law was ridiculed even at the time, and has been at the center of the frequentist / Baysian controversy ever since
- The sun has risen every day for the last 5,000 years. The probability that it **won't** rise tomorrow is:

$$
\begin{aligned}
P(\text{no sun}) &= 1 - P(\text{sun}) \\
&= 1 - \frac{5{,}000 \times 365.25 + 1}{5{,}000 \times 365.25 + 2} \\
&= \frac{1}{1{,}836{,}252}
\end{aligned}
$$

# Laplace’s Law of Succession

- The probability that a 10-year-old will be alive next year is 11/12=0.92, but the probability that a 100-year-old will be alive is 101/102=0.99
- If human cloning is achieved in the lab once, the probability that will be achieved when the experiment is repeated exactly is 2/3.
- Cold fusion has never been achieved in the lab, so the probability that will be achieved on the first try is 1/2.
- These are all misapplications of Laplace’s Law!

# Add-one smoothing

- Laplace's Law gives us a way of dealing with zero counts
- "Add-one" **smoothing** or **discounting**
- If $V$ is the vocabulary size, then

$$\hat{P}_{\text{Laplace}}(w_i|c) = \frac{1 + \text{\# of times word } i \text{ occurs texts in class } c}{V + \text{\# of words in texts in } c}$$

# Lidstone's Law

- Laplace's law assumes a **uniform** prior distribution over models (all coins are equally likely)
- A more general alternative based on a Dirichlet prior, proposed by Hardy (1882) and Lidstone (1920):

$$\hat{P}_{\text{Lidstone}}(w_i|c) = \frac{\alpha + \text{\# of times word } i \text{ occurs texts in class } c}{\alpha V + \text{\# of words in texts in } c}$$

- Johnson (1932) showed that if we set $\mu = N/(N+\alpha V)$, then we can rewrite this as:

$$\mu\, \hat{P}_{\text{MLE}}(w_i|c) + (1-\mu)\frac{1}{V}$$

- This is a **linear interpolation** between the MLE and uniform estimates

# Lidstone's Law

- What value of $\alpha$ should we use?
- A common guess is $\alpha$=1/2 (known as the Jeffreys-Perks Law, the Krichevsky-Trofimov estimator, or Expected Likelihood Estimation)
- Another good one for small $V$ is $\alpha = 1/V$ (Schurmann-Grassberger Law)
- Useful values vary widely, so $\alpha$ could be found using grid search/cross validation
- For a small sample, the prior determines the solution (**hyperparameters**)

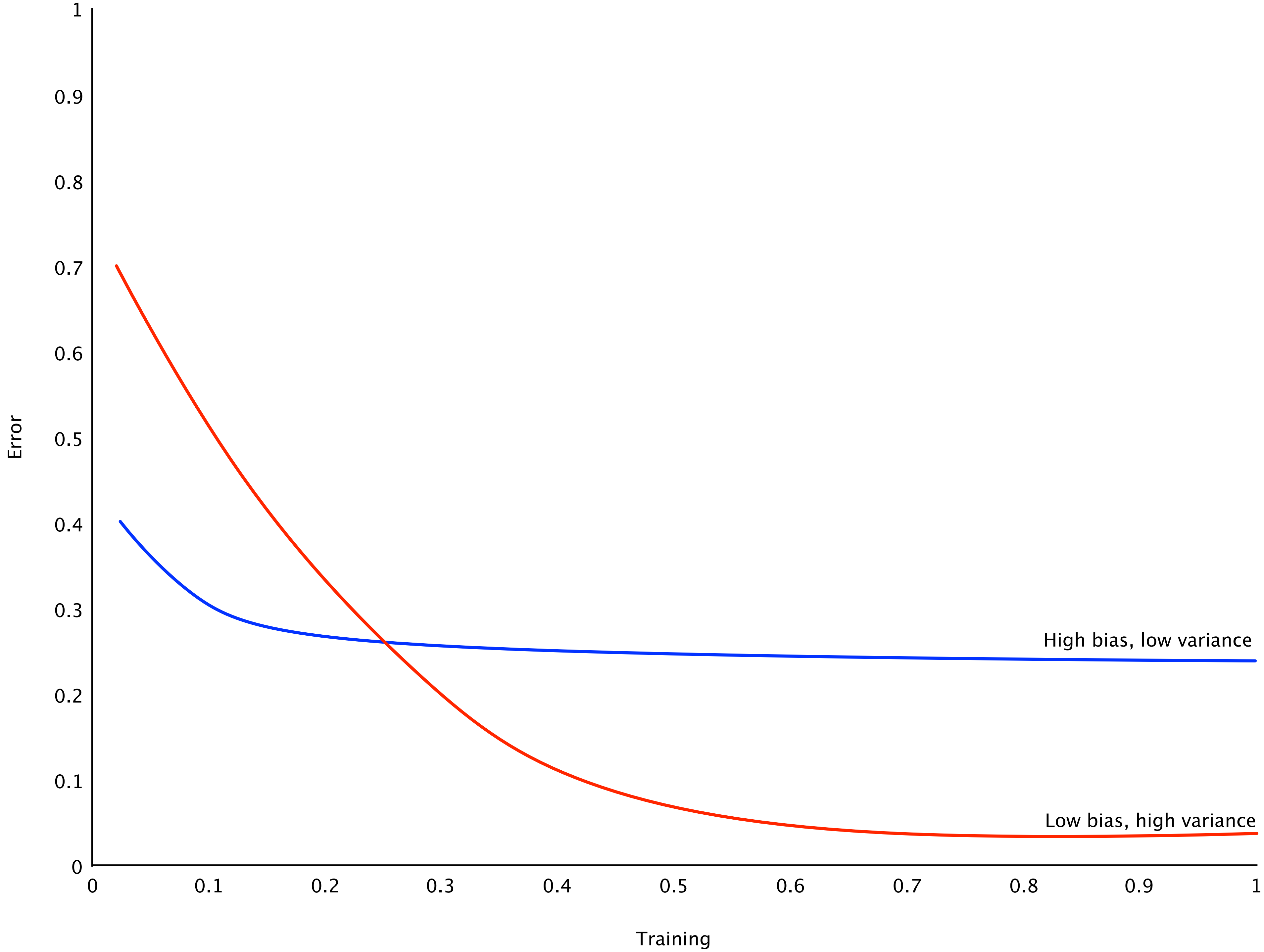
# Bias vs variance

- A general theme in machine learning is a tradeoff between prior information and properties of the training data
- We need prior assumptions about the solution in order to generalize, but if incorrect this can lead to errors
- We can formalize this (Geman 1992):

$$\text{error} = \text{noise} + \text{bias} + \text{variance}$$

- Bias comes from the nature of the learning algorithm
- Variance comes from properties of the training data

# Bias vs variance

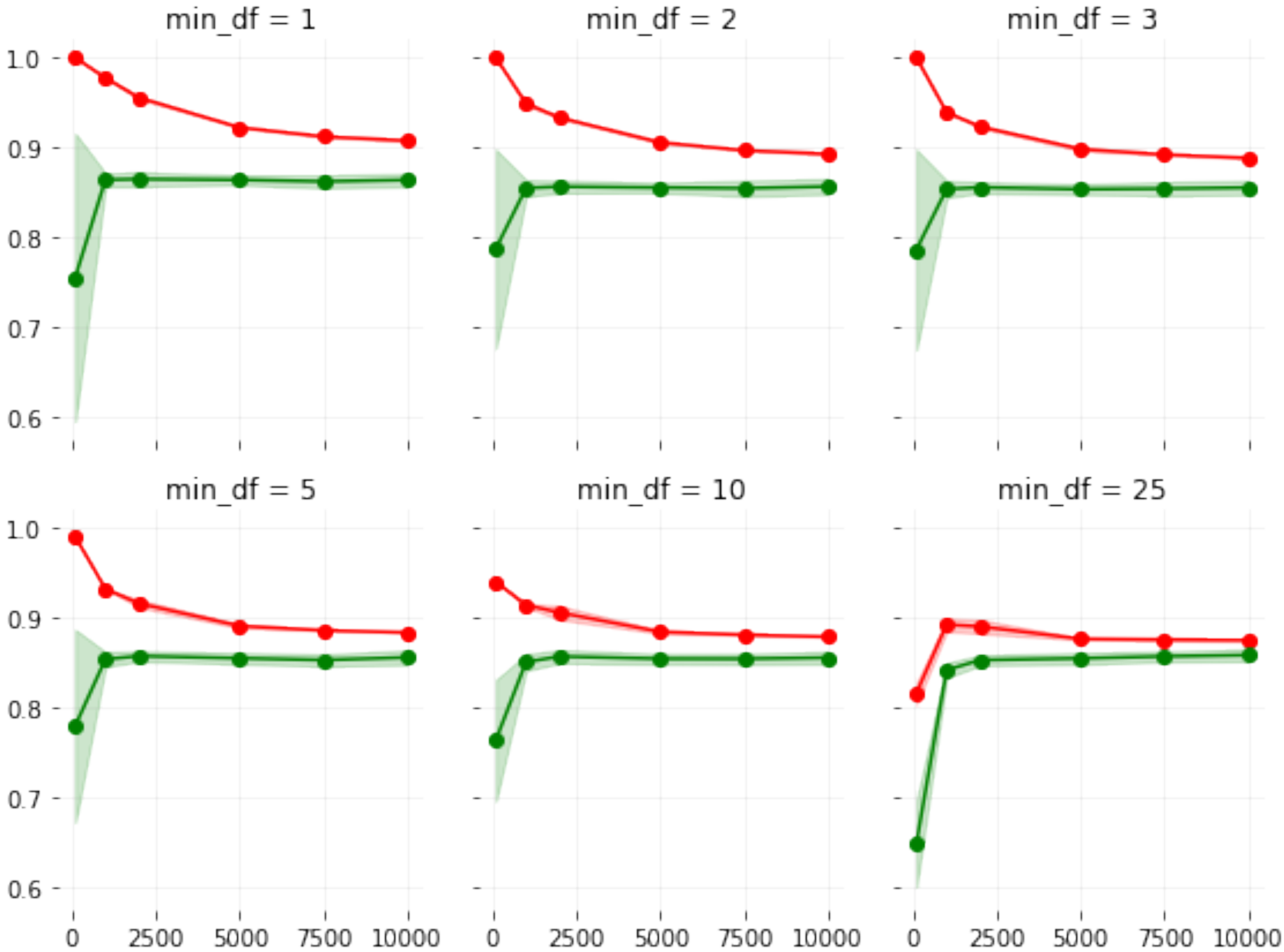


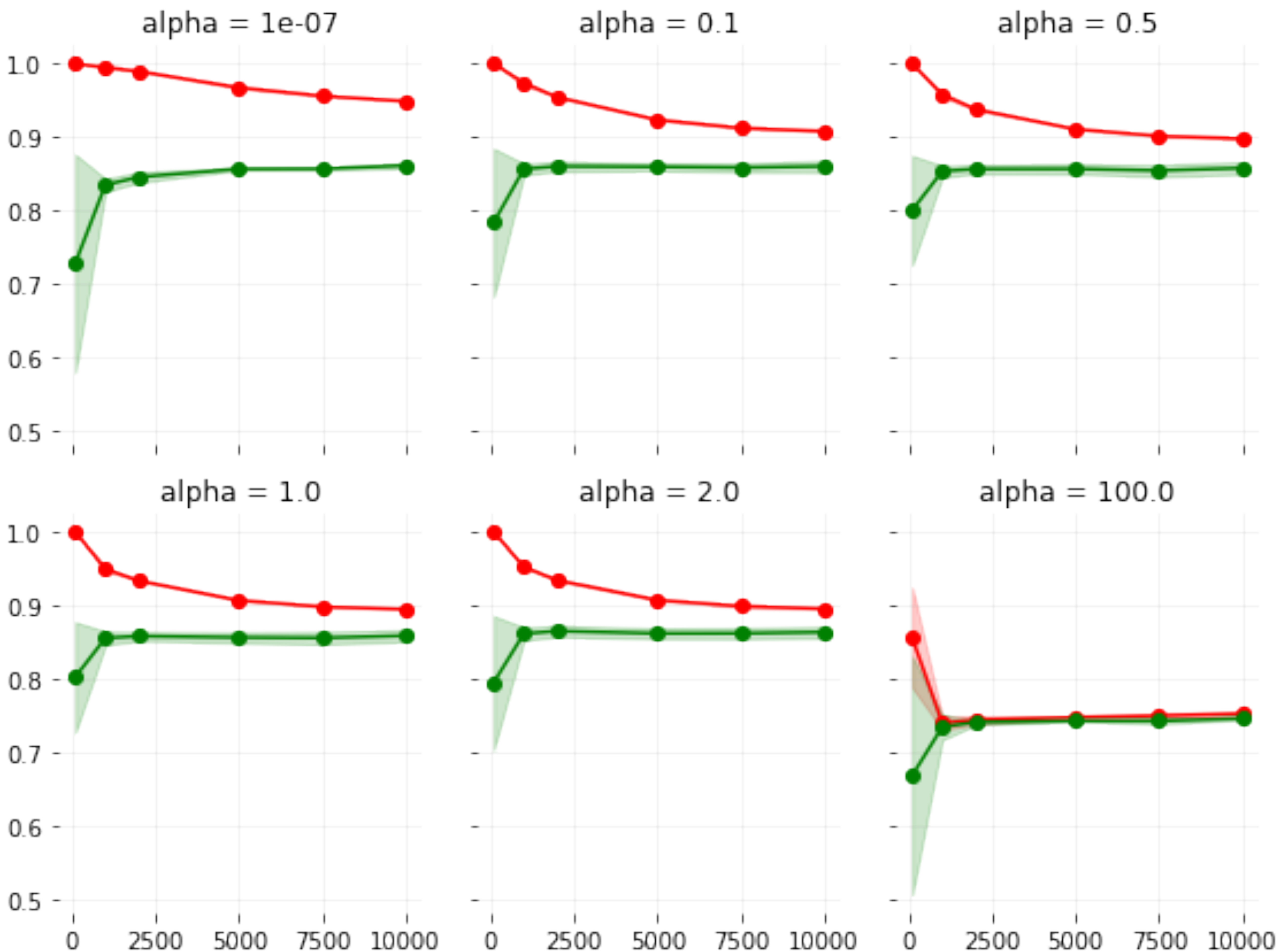
# Bias vs variance

- Different methods provide different types of bias, either implicitly or explicitly
- Appropriate bias reduces variance
- Inappropriate bias reduces variance, but increases bias error
- When we don’t have enough training data, variance is more of a problem than bias error (Curse of Dimensionality!)
- Deliberately increasing bias (e.g., by smoothing, lower casing, pruning words from vocabulary, etc.) can reduce variance enough that overall error drops

# Bias vs variance

- Compare learning curves for MultinomialNB classifier trained on RCV1 politics articles
- Calculate training and test accuracy for varying number of articles in training set (variance)
- Compare curves for different values of hyperparameters (bias)


min_df = 1
min_df = 2
min_df = 3
min_df = 5
min_df = 10
min_df = 25
1.0
0.9
0.8
0.7
0.6
0
2500
5000
7500
10000



alpha = 1e-07
alpha = 0.1
alpha = 0.5
alpha = 1.0
alpha = 2.0
alpha = 100.0
1.0
0.9
0.8
0.7
0.6
0.5
0
2500
5000
7500
10000

# Linear classifiers

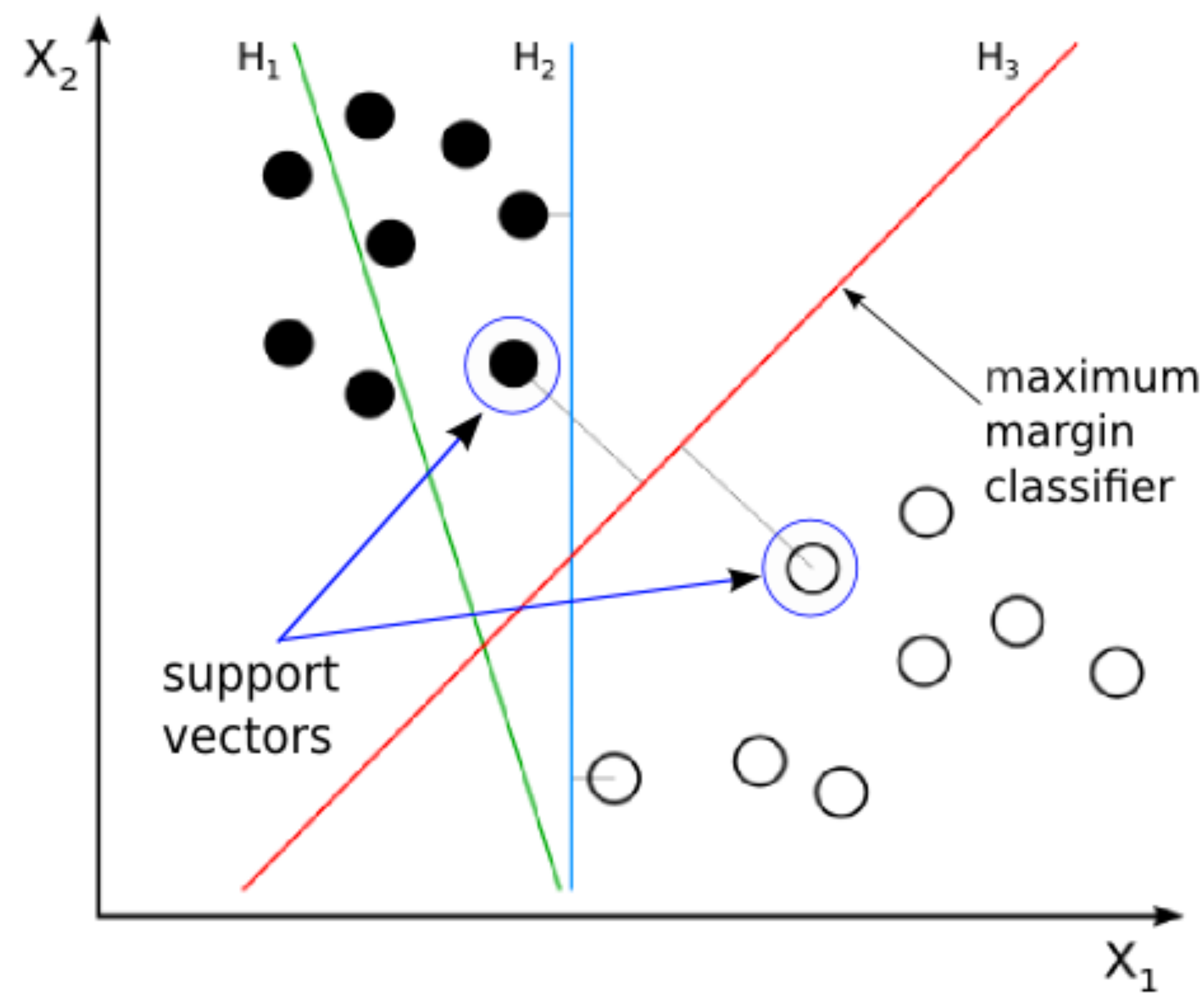
- Naive Bayes classifiers learn a **linear decision function**

$$\log P(x, c) \propto \log P(c) + \sum_i \log P(x_i|c)$$

- Logistic regression (MaxEnt) is another linear classifier
  - $C$ is a smoothing parameter (smaller values of $C$ mean more smoothing)
  - Two smoothing methods: $L_2$ and $L_1$
    - $L_2$ often gives more accurate results
    - $L_1$ prefers sparse models that only include the most predictive words

# Linear classifiers

- Linear Support Vector Machines maximize the margin
- Represent decision boundary via texts (support vectors) rather than words
- Scale to very large vocabularies, but not large numbers of documents



# Model comparison

- Best `LogisticRegression` model ($C$=0.01, min_df=1, max_df=0.25) accuracy is **89.20%**
- Best `LinearSVC` model ($C$=0.001, min_df=1 max_df=0.25) accuracy is **89.11%**
- These are 10-CV averages — can we expect LR better than SVC on new data?
- Wilcoxon signed-rank test on paired scores

# Model comparison

```
In [92]: from scipy.stats import wilcoxon

In [93]: folds = StratifiedKFold(shuffle=True, n_splits=10, random_state=10)

In [94]: lr_model.set_params(**lr_grid_search.best_params_)
         lr_score = cross_val_score(lr_model, df['tokens'], df['politics'], cv=folds, n_jobs=-1)

In [95]: svm_model.set_params(**svm_grid_search.best_params_)
         svm_score = cross_val_score(svm_model, df['tokens'], df['politics'], cv=folds, n_jobs=-1)

In [96]: lr_score.mean(), svm_score.mean()

Out[96]: (0.8920, 0.8911)

In [97]: lr_score - svm_score

Out[97]: array([ 0.0047, -0.0007, -0.004 , -0.0013,  0.0033,  0.0013,  0.0007,
                 0.004 , -0.0013,  0.0027])

In [98]: wilcoxon(lr_score, svm_score, correction=True)

Out[98]: WilcoxonResult(statistic=18.5, pvalue=0.38596207926442694)
```

# Feature transforms

- CountVectorizer converts a document into a vector of word counts
- Tf-Idf transform scales counts to give more weight to words that are more likely to be interesting
  - $\text{tf}(w, d)$ =# of times $w$ occurs in $d$
  - $\text{df}(w)$ = # of documents that contain $w$
  - D = total # of documents

$$\begin{aligned}\text{tf-idf}(w, d) &= \text{tf}(w, d) \times \text{idf}(w) \\ &= \text{tf}(w, d) \times \log \frac{D}{1 + \text{df}(w)}\end{aligned}$$

# Text classification

- Reuters texts are easy to classify
  - Large number of training examples
  - Long texts
  - Formal edited language
  - Consistent labeling
- That’s not real life . . .