

Statistical Data Mining and Machine Learning

Hilary Term 2016

Dino Sejdinovic
Department of Statistics
Oxford

Slides and other materials available at:
<http://www.stats.ox.ac.uk/~sejdinov/sdmml>

Performance Evaluation

Example: Spam Dataset

A data set collected at Hewlett-Packard Labs, that classifies 4601 e-mails as spam or non-spam. 57 variables indicate the frequency of certain words and characters.

```
> library(kernlab)
> data(spam)
> dim(spam)
[1] 4601 58
> spam[1:2,]
  make address all num3d our over remove internet order mail receive will
1 0.00 0.64 0.64 0 0.32 0.00 0.00 0.00 0 0.00 0.00 0.64
2 0.21 0.28 0.50 0 0.14 0.28 0.21 0.07 0 0.94 0.21 0.79
  people report addresses free business email you credit your font num000
1 0.00 0.00 0.00 0.32 0.00 1.29 1.93 0 0.96 0 0.00
2 0.65 0.21 0.14 0.14 0.07 0.28 3.47 0 1.59 0 0.43
  money hp hpl george num650 lab labs telnet num857 data num415 num85
1 0.00 0 0 0 0 0 0 0 0 0 0 0
2 0.43 0 0 0 0 0 0 0 0 0 0 0
  technology num1999 parts pm direct cs meeting original project re edu table
1 0 0.00 0 0 0 0 0 0 0 0 0 0
2 0 0.07 0 0 0 0 0 0 0 0 0 0
  conference charSemicolon charRoundbracket charSquarebracket charExclamation
1 0 0 0.000 0 0.778
2 0 0 0.132 0 0.372
  charDollar charHash capitalAve capitalLong capitalTotal type
1 0.00 0.000 3.756 61 278 spam
2 0.18 0.048 5.114 101 1028 spam
> str(spam$type)
Factor w/ 2 levels "nonspam","spam": 2 2 2 2 2 2 2 2 2 2 ...
```

Spam Dataset

Use logistic regression to predict spam/not spam.

```
## let Y=0 be non-spam and Y=1 be spam.  
Y <- as.numeric(spam$type)-1  
X <- spam[ , -ncol(spam)]  
  
gl <- glm(Y ~ ., data=X, family=binomial)
```

Spam Dataset

How good is the classification?

```
> table(spam$type)
nonspam    spam
   2788    1813

> proba <- predict(gl,type="response")
> predicted_spam <- as.numeric( proba>0.5)
> table(predicted_spam,Y)
      Y
predicted_spam  0    1
               0 2666 194
               1  122 1619

> predicted_spam <- as.numeric( proba>0.95)
> table(predicted_spam,Y)
      Y
predicted_spam  0    1
               0 2766  810
               1   22 1003
```

Advantage of a probabilistic approach: predictive probabilities give interpretable confidence to predictions. Soft classification rules for other classifiers, e.g., support vector machines can be poorly calibrated if we are to interpret them as probabilities.

Spam Dataset

- We are viewing the prediction error on the training set. Not necessarily representative of the generalization ability.
- Separate in training and test set 50/50.

```
n <- length(Y)
train <- sample( n, round(n/2) )
test<-(1:n)[-train]
```

- Fit only on training set and predict on both training and test set.

```
gl <- glm(Y[train] ~ ., data=X[train,],family=binomial)

proba_train <- predict(gl,newdata=X[train,],type="response")
proba_test  <- predict(gl,newdata=X[test,],type="response")
```

Spam Dataset

Results for training and test set:

```
> predicted_spam_lr_train <- as.numeric(proba_train > 0.95)
> predicted_spam_lr_test  <- as.numeric(proba_test  > 0.95)
```

```
> table(predicted_spam_lr_train, Y[train])
predicted_spam_lr_train    0    1
                        0 1401  358
                        1    8  533
```

```
> table(predicted_spam_lr_test, Y[test])
predicted_spam_lr_test    0    1
                        0 1357  392
                        1   22  530
```

Note: testing performance is worse than training performance.

Spam Dataset

Compare with LDA.

```
library(MASS)
lda_res <- lda(x=X[train,], grouping=Y[train])

proba_lda_test <- predict(lda_res, newdata=X[test,])$posterior[,2]
predicted_spam_lda_test <- as.numeric(proba_lda_test > 0.95)

> table(predicted_spam_lr_test, Y[test])
predicted_spam_lr_test    0    1
                        0 1357  392
                        1   22  530

> table(predicted_spam_lda_test, Y[test])
predicted_spam_lda_test    0    1
                        0 1361  533
                        1   18  389
```

- LDA has a larger number of false positives but a smaller number of false negatives.
 - Above results are for a single threshold (0.95) - how to keep track of what happens across multiple thresholds?
 - More generally, how to compare the classifiers fairly when the number of positive and negative examples is very different?

Performance Measures

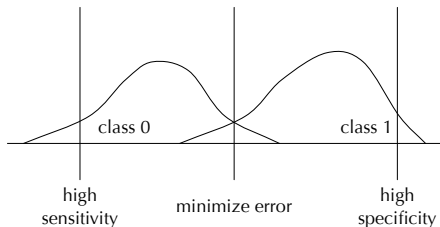
- **Confusion matrix:**

True state		0	1
Prediction	0	# true negative	# false negative
	1	# false positive	# true positive

- **Accuracy:** $(TP + TN)/(TP + TN + FP + FN)$.
- **Error rate:** $(FP + FN)/(TP + TN + FP + FN)$.
- **Sensitivity (true positive rate):** $TP/(TP + FN)$.
- **Specificity (true negative rate):** $TN/(TN + FP)$.
- **False positive rate (1-Specificity):** $FP/(TN + FP)$.
- **Precision:** $TP/(TP + FP)$.
- **Recall** (same as Sensitivity): $TP/(TP + FN)$.
- **F1:** harmonic mean of precision and recall.

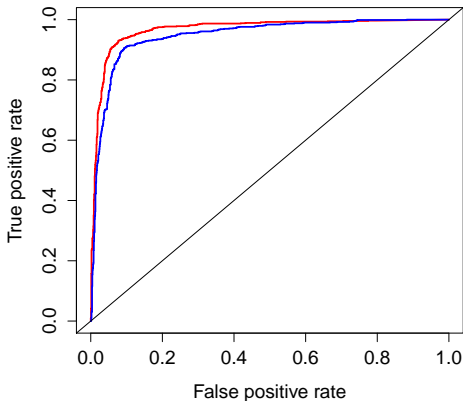
- As we vary the prediction threshold c from 0 to 1:

- Specificity varies from 0 to 1.
- Sensitivity goes from 1 to 0.



ROC (Receiver Operating Characteristic) Curves

ROC curve: plot TPR (sensitivity) vs FPR (1-specificity). LDA = blue; LR = red.



LR beats LDA on this dataset in terms of the **area under ROC (AUC)**: **probability that the classifier will score a randomly drawn positive example higher than a randomly drawn negative example**. Also called Wilcoxon-Mann-Whitney statistic.

ROC Curves

R library `ROCR` contains various performance measures, including AUC.

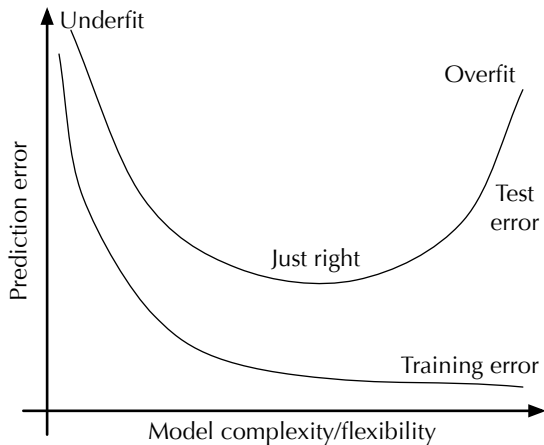
```
> library(ROCR)
> pred_lr <- prediction(proba_test, Y[test])
> perf <- performance(pred_lr, measure = "tpr", x.measure = "fpr")
> plot(perf, col='red', lwd=2)
> pred_lda <- prediction(proba_lda_test, Y[test])
> perf <- performance(pred_lda, measure = "tpr", x.measure = "fpr")
> plot(perf, col='blue', add=TRUE, lwd=2)
> abline(a=0, b=1)
> auc_lda <- as.numeric(performance(pred_lda, "auc")@y.values)
> auc_lda
[1] 0.9472542
> auc_lr <- as.numeric(performance(pred_lr, "auc")@y.values)
> auc_lr
[1] 0.9673279
```

Validation and Model Selection

Generalization

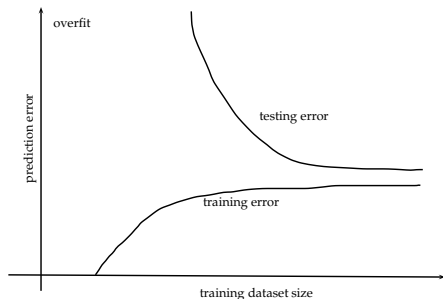
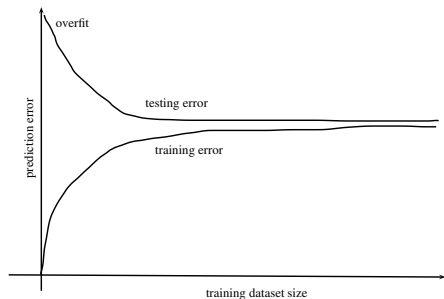
- Generalization ability: what is the out-of-sample error of learner f ?
- training error \neq testing error.
- We learn f by minimizing some variant of empirical risk $R^{\text{emp}}(f)$ - what can we say about the true risk $R(f)$?
- Two important factors determining generalization ability:
 - Model complexity
 - Training data size

Learning Curves



Fixed dataset size, varying model complexity.

Learning Curves



Fixed model complexity, varying dataset size.

Two models: one simple, one complex. Which is which?

Bias-Variance Tradeoff

- Where does the prediction error come from?
- Example: Squared loss in regression: $\mathcal{X} = \mathbb{R}^p$, $\mathcal{Y} = \mathbb{R}$,

$$L(Y, f(X)) = (Y - f(X))^2$$

- Optimal f is the conditional mean:

$$f_*(x) = \mathbb{E}[Y|X = x]$$

- Follows from $R(f) = \mathbb{E}_X \mathbb{E} \left[(Y - f(X))^2 \mid X \right]$ and

$$\begin{aligned} & \mathbb{E} \left[(Y - f(X))^2 \mid X = x \right] \\ &= \mathbb{E} \left[Y^2 \mid X = x \right] - 2f(x) \mathbb{E} \left[Y \mid X = x \right] + f(x)^2 \\ &= \text{Var} \left[Y \mid X = x \right] + (\mathbb{E} \left[Y \mid X = x \right] - f(x))^2. \end{aligned}$$

Bias-Variance Tradeoff

- Optimal risk is the intrinsic conditional variability of Y (noise):

$$R(f_*) = \mathbb{E}_X [\text{Var} [Y|X]]$$

- **Excess risk:**

$$R(f) - R(f_*) = \mathbb{E}_X \left[(f(X) - f_*(X))^2 \right]$$

- How does the excess risk behave **on average**?
- Consider a randomly selected dataset $\mathcal{D} = \{(X_i, Y_i)\}_{i=1}^n$ and $f^{(\mathcal{D})}$ trained on \mathcal{D} using a model (hypothesis class) \mathcal{H} .

$$\begin{aligned} \mathbb{E}_{\mathcal{D}} \left[R(f^{(\mathcal{D})}) - R(f_*) \right] &= \mathbb{E}_{\mathcal{D}} \mathbb{E}_X \left[\left(f^{(\mathcal{D})}(X) - f_*(X) \right)^2 \right] \\ &= \mathbb{E}_X \mathbb{E}_{\mathcal{D}} \left[\left(f^{(\mathcal{D})}(X) - f_*(X) \right)^2 \right]. \end{aligned}$$

Bias-Variance Tradeoff

- Denote $\bar{f}(x) = \mathbb{E}_{\mathcal{D}} f^{(\mathcal{D})}(x)$ (average decision function over all possible datasets)

$$\mathbb{E}_{\mathcal{D}} \left[\left(f^{(\mathcal{D})}(X) - f_*(X) \right)^2 \right] = \underbrace{\mathbb{E}_{\mathcal{D}} \left[\left(f^{(\mathcal{D})}(X) - \bar{f}(X) \right)^2 \right]}_{\text{Var}_X(\mathcal{H}, n)} + \underbrace{\left(\bar{f}(X) - f_*(X) \right)^2}_{\text{Bias}_X^2(\mathcal{H}, n)}$$

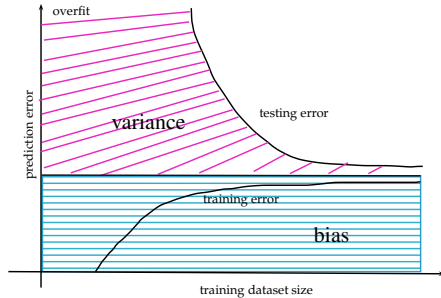
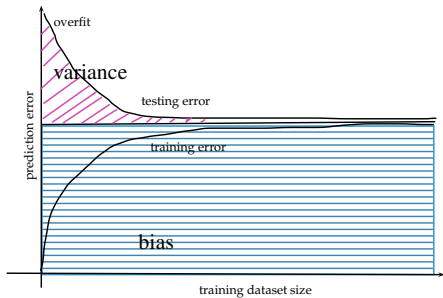
Now,

$$\mathbb{E}_{\mathcal{D}} R(f^{(\mathcal{D})}) = R(f_*) + \mathbb{E}_X \text{Var}_X(\mathcal{H}, n) + \mathbb{E}_X \text{Bias}_X^2(\mathcal{H}, n)$$

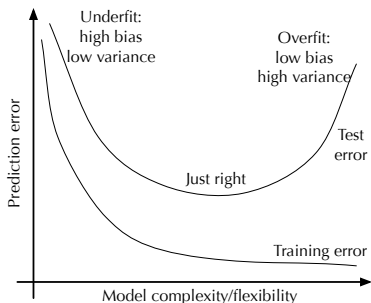
Where does the prediction error come from?

- Noise:** Intrinsic difficulty of regression problem.
- Bias:** How far away is the best learner in the model (average learner over all possible datasets) from the optimal one?
- Variance:** How variable is our learning method if given different datasets?

Learning Curves



Building models to trade bias with variance



- Building a machine learning model involves trading between its bias and variance.
 - Bias reduction at the expense of a variance increase: building more complex models, e.g. adding nonlinear features and additional parameters, increasing the number of hidden units in neural nets, using decision trees with larger depth.
 - Variance reduction at the expense of a bias increase: increasing the regularization parameter, early stopping, using k-nearest neighbours with larger k.

Validation and Cross-Validation

Empirical vs True Risk

- In general,

$$R(f) = R^{\text{emp}}(f) + \text{overfit penalty.}$$

- Overfit penalty depends on the complexity of the model (**VC analysis**).
- Regularization: **approximate the overfit penalty**. More complex the model, larger the overfit penalty.
- (Cross-)Validation: try to **estimate $R(f)$ directly**.
- For any example not used in training:

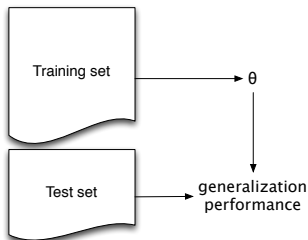
$$\mathbb{E} [L (y_{\text{test}}, f(x_{\text{test}}))] = R(f).$$

- But for examples used in training:

$$\mathbb{E} [L (y_{\text{train}}, f(x_{\text{train}}))] \neq R(f).$$

Optimizing Tuning Parameters

- How can we optimize generalization ability, via optimizing choice of tuning parameters, model size, and learning parameters?
- Suppose we have split data into training/test set.
- Test set can be used to determine generalization ability, and used to choose best setting of tuning parameters/model size/learning parameters with best generalization.
- Once these tuning parameters are chosen, still important to determine generalization ability, but cannot use performance on test set to gauge this anymore!
- Idea: split data into 3 sets: training set, test set, and **validation set**.



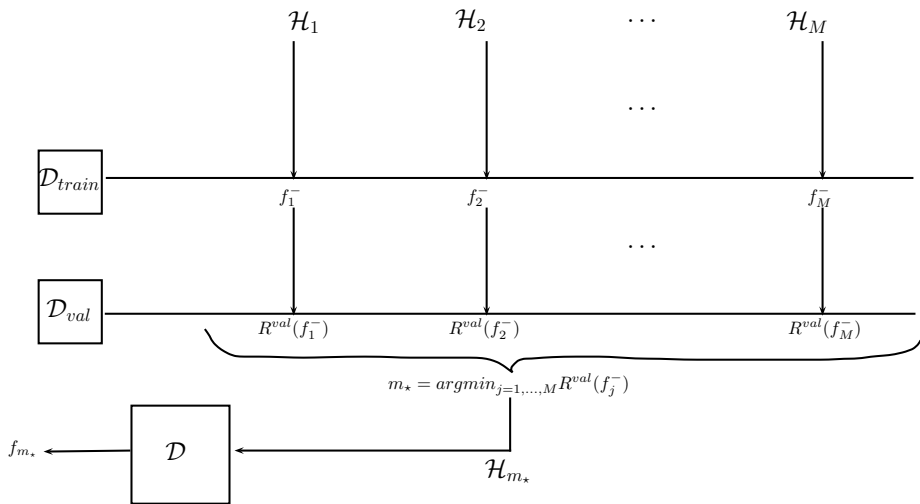
Validation error

- **Out-of-sample average loss.** For a dataset $\{\tilde{x}_i, \tilde{y}_i\}_{i=1}^v$ unseen in training

$$R^{\text{val}}(f) = \frac{1}{v} \sum_{i=1}^v L(\tilde{y}_i, f(\tilde{x}_i))$$

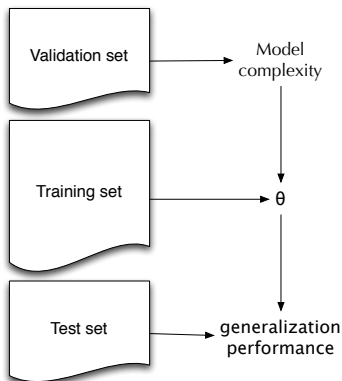
- $\mathbb{E} [R^{\text{val}}(f)] = R(f)$, $\text{Var} [R^{\text{val}}(f)] \asymp \frac{1}{v}$, i.e. $R^{\text{val}}(f) = R(f) \pm \mathcal{O}\left(\frac{1}{\sqrt{v}}\right)$
- Just like testing error so far.
- It becomes validation error only once it is used to **change our learning**.

Validation



Validation

- For each combination of tuning parameters γ :
 - Train our model on the training set, fit parameters $\theta = \theta(\gamma)$, obtaining decision function $f_{\theta(\gamma)}$.
 - Evaluate $R^{\text{val}}(f_{\theta(\gamma)})$ average loss on a validation set.
- Pick γ^* with best performance on validation set.
- Using γ^* , train on both training and validation set to obtain the optimal θ^* .
- $R^{\text{val}}(f_{\theta(\gamma^*)})$ is now a **biased estimate** of $R(f_{\theta(\gamma^*)})$ and can be overly optimistic!
- Evaluate model with γ^*, θ^* on test set, reporting generalization performance.



Bias introduced by validation

- **Example:** Selecting between two equally bad classifiers f_1 and f_2 :

$$R(f_1) = R(f_2) = 0.5.$$

- Assume that we have independent unbiased estimators $R_1 = R^{\text{val}}(f_1)$, $R_2 = R^{\text{val}}(f_2)$, both uniform on $[0, 1]$
- Learning rule f_* chosen to minimize R^{val} is either f_1 or f_2 , so also equally bad.
- But $\mathbb{E} \min\{R_1, R_2\} = \frac{1}{3}$, so in terms of validation error it may appear that we are getting an improvement!

Validation error and Generalization

How contaminated are different parts of data in terms of being able to tell us something about generalization ability?

- Training data: fully contaminated, used in learning - $R^{\text{emp}}(f)$ is usually far from $R(f)$ (unless the model is too simple for the amount of data).
- Validation data: partly contaminated, used in model selection / meta-learning - $R^{\text{val}}(f)$ is biased, but still useful, provided that:
 - we have a large enough validation set size v
 - we do not use it to select from a large number M of models
- Test data: clean, not used for any part of learning.

Typically,

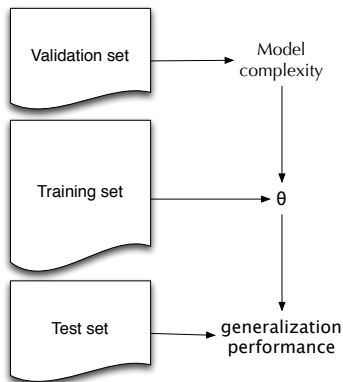
$$R(f) \leq R^{\text{val}}(f) + \underbrace{\mathcal{O}\left(\sqrt{\frac{\log M}{v}}\right)}_{\text{overfit penalty of the meta-model}}$$

Size of validation set?

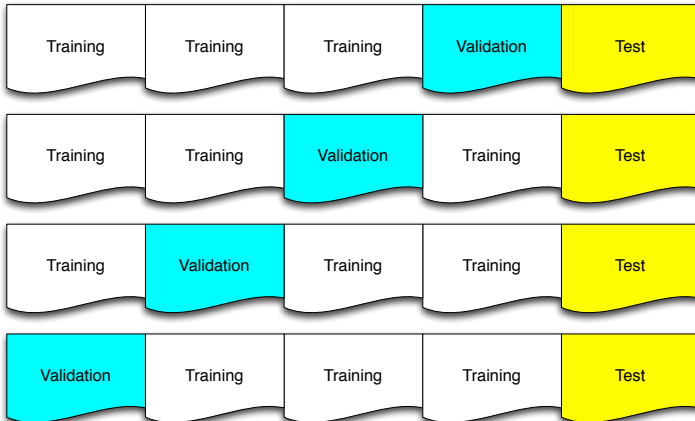
- In practice, there is just one dataset! If v is used for computing validation error, then only $n - v$ used for training.
 - Small v : $R^{\text{val}}(f^-)$ is a bad estimate of $R(f^-)$
 - Large v : $R^{\text{val}}(f^-)$ is a reliable estimate of a much worse risk (f^- learned on much less data than f)!
- We are using:

$$R(f) \underset{\text{(need small } v)}{\approx} R(f^-) \underset{\text{(need large } v)}{\approx} R^{\text{val}}(f^-)$$

- Wasteful to split into 3 subsets.
- Different approach: **cross-validation**.



Cross-Validation



Cross-Validation

- Basic approach:

- Split training set into T folds.
- For each γ and each $t = 1, \dots, T$:
 - Use fold t as validation set and the rest to train the model parameters $\theta_t = \theta_t(\gamma)$, obtaining decision function $f_{t,\gamma}^-$.

$$R_t^{\text{val}}(f_{t,\gamma}^-) = \frac{1}{|\text{Fold}(t)|} \sum_{i \in \text{Fold}(t)} L(y_i, f_{t,\gamma}^-(x_i))$$

- Choose γ^* which minimizes validation error averaged over folds

$$\frac{1}{T} \sum_{t=1}^T R_t^{\text{val}}(f_{t,\gamma}^-)$$

- Train model with tuning parameter γ^* on all training set to obtain f_{γ^*} .
- Report generalization performance on test set.
- **Leave-One-Out (LOO)** cross validation: one data item per fold, i.e., $T = n$.

Cross-validation can be computationally expensive ($T \times$ increase in complexity).

Leave-One-Out Cross-Validation

Leave-one-out (LOO) cross validation: one data item per fold, i.e., $T = n$.

- Since only one data item not used in training, $R(f_{t,\gamma}^-)$ are all very close to $R(f_\gamma)$ (small v benefit).
- Thus,

$$\frac{1}{n} \sum_{t=1}^n R_t^{\text{val}}(f_{t,\gamma}^-) = \frac{1}{n} \sum_{t=1}^n L(y_t, f_{t,\gamma}^-(x_t))$$

has a small variance (large v benefit).

- All examples for validation and all examples for training.
- **summands are no longer independent**