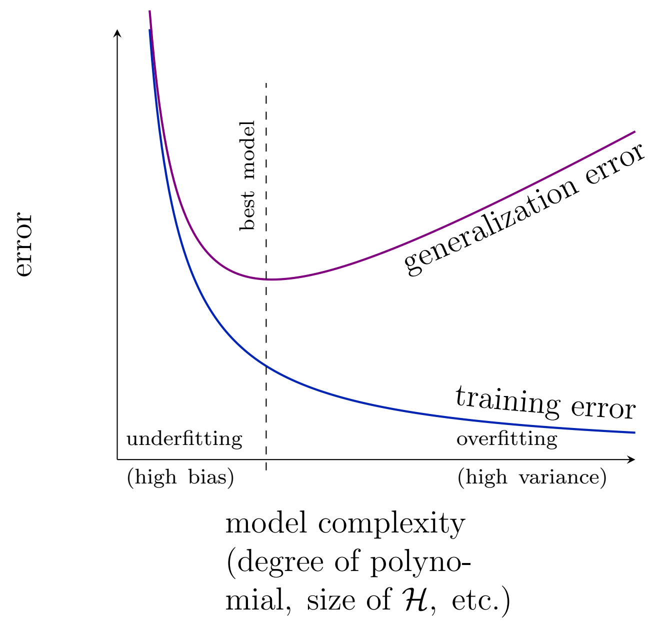
Lecture 9

1. Model selection
2. Feature selection
3. Bayesian approach and regularization
4. Online learning
5. Advice for apply ML algorithms

1 Model selection

In previous lecture we showed the trade-off between bias and variance as follows



The main question is how to choose the model complexity: it could be degree of polynomial, bandwidth parameter in locally weighted linear regression or parameter C in SVM. Assuming that we have list of models

$$\mathcal{M} = \{M_1, M_2, M_3, \ldots\},$$

the wrong way would be to choose the model with the lowest error on the training set, because in such a way the most complicated model will be chosen. Consider different techniques to choose the best model (as before let S be our training set).

- **Hold-out cross validation**

    1. Split S into S_{train} (70%) and S_{val} (30%).

2. Train each model $M \in \mathcal{M}$ on S_{train} and predict on S_{val}.
3. Pick the model $M \in \mathcal{M}$ with the lowest error on S_{val}.

- **K-fold cross validation**

  1. Split S in K pieces (**folds**).
  2. Train on $K-1$ folds and predict on the remaining fold. Average the accuracy for all K folds.
  3. Pick the model $M \in \mathcal{M}$ with the lowest average error.

- **Leave-one-out cross validation**

  This technique is similar to K-fold cross validation but number of folds K equals to the size of the training set S. This method is very computationally expensive.

1.1 Python implementation

Import necessary libraries:

```
In [1]: import numpy as np
        import pandas as pd
        import random
        import math
        import sklearn.datasets as ds
        %matplotlib inline
```

Cross validation is one of the most important procedure, when you select the model or tune the parameters. There are several possibilities to perform cross validation in the `scikit-learn`.

```
In [2]: from sklearn.cross_validation import train_test_split
        from sklearn.cross_validation import StratifiedKFold
        from sklearn.cross_validation import LeaveOneOut
```

The simplest way to build cross validation splits is to use the method `train_test_split`:

```
In [3]: boston = ds.load_boston()
        X = boston.data
        y = boston.target/50.
        X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                            test_size=0.33,
                                                            random_state=42)
```

```
In [4]: print 'Number of training examples: ' + str(X_train.shape[0])
        print 'Number of validation examples: ' + str(X_test.shape[0])
```

```
Number of training examples: 339
Number of validation examples: 167
```

Another way is to use K-fold or leave-one-out cross validation procedures. Notice that we use the method `StratifiedKFold` that guarantees that the distribution of target variable stays the same for different folds.

```
In [5]: skf = StratifiedKFold(y, n_folds=3, shuffle=True, random_state=21387)
        for train_index, test_index in skf:
            X_train = X[train_index]
            X_test = X[test_index]
            print 'Number of training examples: ' + str(X_train.shape[0])
            print 'Number of validation examples: ' + str(X_test.shape[0])
```

```
Number of training examples: 323
Number of validation examples: 183
Number of training examples: 332
Number of validation examples: 174
Number of training examples: 357
Number of validation examples: 149
```

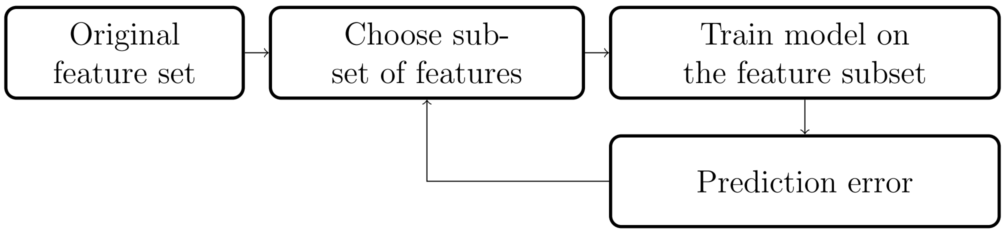
Using the method `LeaveOneOut` we can perform leave-one-out cross validation procedure in similar way.

2 Feature selection

Another approach to model selection is called feature selection. For many machine learning problems feature space is a very high-dimensional. To reduce it we should choose the subset of features: if we have n features then the number of possible subsets is 2^n. Feature selection methods can be splitted in three groups: wrappers, filters and embedded methods.

- Wrappers.

  Wrappers are model-based methods for feature selection and are considered to be the most effective and computationally intractable algorithms. The main principle is shown on the picture.



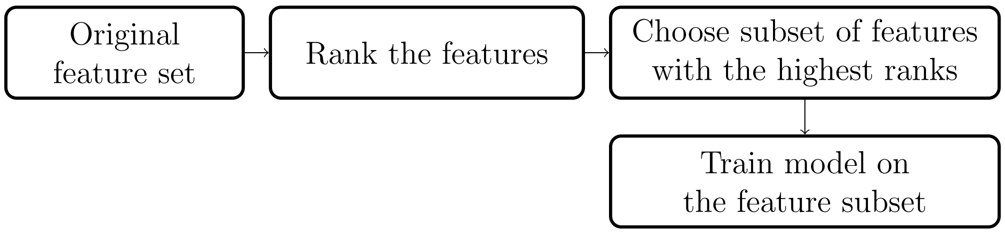
To find the most relevant and informative subset of features, the model is trained for different subsets of features. The subset with the lowest error is selected as best set of features. Because wrapper methods utilize the model algorithm to extract relationships and causality between features, they are considered more effective and hence more desired than filters and embedded methods. To increase the speed of wrapper methods we can use different search methods such as forward selection, backward elimination and stepwise regression are available.

- **Forward selection** begins with no variables and progressively adds features until maximum reduction in prediction error is reached.
- **Backward elimination** begins with all features and progressively removes features having smallest contributions.
- **Stepwise selection** starts by adding features until reaching some stopping criteria. Then the algorithm starts dropping features until reaching another stopping criteria and so on.

Notice that these techniques do not guarantee the selection of the global optimal feature set.

- Filters.

Filters evaluate feature importance as a preprocessing operation to model training as depicted in the figure.


Original feature set
Rank the features
Choose subset of features with the highest ranks
Train model on the feature subset

The main difference between filters and wrappers is that filters do not use the training procedure to capture the relationship between features. Rather, they use some information metric to calculate feature ranking from the data without direct input from the target. Popular information metrics include t-statistic, p-value, Pearson correlation coefficient and other correlation measures. One more example of such metric is mutual information

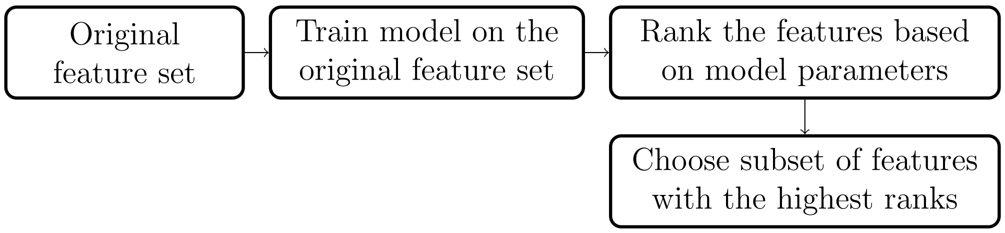
$$MI(x_i, y) = \sum_{x_i \in \{0,1\}} \sum_{y \in \{0,1\}} p(x_i, y) \ln \frac{p(x_i, y)}{p(x_i)p(y)} = KL(p(x_i, y) || p(x_i)p(y)).$$

When all features are ranked the top k features are picked (k is chosen with the cross validation procedure).

Computationally, filters are more efficient than wrappers as they require only the computation of n scores for n features. They are also more robust against overfitting than wrappers.

- Embedded methods.

  Embedded methods use training procedure to obtain feature rankings:


Original feature set
Train model on the original feature set
Rank the features based on model parameters
Choose subset of features with the highest ranks

  Regularization methods are the most common forms of embedded methods.

3 Bayesian approach and regularization

In previous lectures we have maximized the likelihood to get coefficients for the model, for example, for linear regression the maximum likelihood is

$$\theta = \arg\max_{\theta} \prod_{i=1}^{m} p(y^{(i)} \,|\, x^{(i)}, \theta).$$

In this technique we do not assume anything about parameters θ. Bayesian approach contains additional step: we add prior on θ, for example,

$$\theta \sim N(0, \tau^2 I),$$

where τ^2 is a vector of variances and I is a unit matrix. The Bayes rule gives

$$p(\theta \,|\, S) = \frac{p(S \,|\, \theta) p(\theta)}{p(S)},$$

where $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ is a training set. This formula can be transformed to

$$p(\theta \,|\, S) \propto \left(\prod_{i=1}^{m} p(y^{(i)} \,|\, x^{(i)}, \theta) \right) p(\theta), \tag{1}$$

because

$$p(S \,|\, \theta) = \prod_{i=1}^{m} p((x^{(i)}, y^{(i)}) \,|\, \theta) = \prod_{i=1}^{m} p(y^{(i)} \,|\, x^{(i)}, \theta) \cdot p(x^{(i)} \,|\, \theta).$$

Left side of the formula (1) is called **a “posterior”**. For the new test example (x, y) we obtain

$$p(y \,|\, x, S) = \int_{\theta} p(y \,|\, x, \theta) \cdot p(\theta \,|\, S) \, d\theta$$

(here we treat θ as a random variable, that's why we put comma instead of semicolon) and in particularly

$$E\left[y \,|\, x, S\right] = \int\limits_y y \cdot p(y \,|\, x, S)\, dy.$$

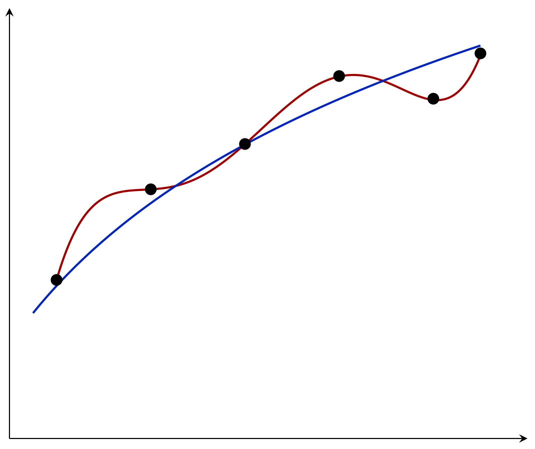
For many problems last two steps are computationally difficult, because usually, θ is many dimensional vector and the integration in $\mathbb{R}^n$ is not simple task. Easier way is to calculate

$$\hat{\theta}_{MAP} = \arg\max_\theta p(\theta \,|\, S) = \arg\max_\theta \left(\prod_{i=1}^m p(y^{(i)} \,|\, x^{(i)}, \theta)\right) p(\theta)$$

where "MAP" stands for "maximum a posteriori". After we found the set of parameters the prediction is calculated as before:

$$h_{\hat{\theta}_{MAP}}(x) = \hat{\theta}^T_{MAP} x.$$

The intuition behind the Bayesian technique is the following: if $\theta \sim N(0, \tau^2 I)$, then almost all parameters should be close to zero. Applying Bayesian regularization (blue line on the picture) will lead to reducing high variance compared to the model without regularization (red line on the picture):



To compare models without and with regularization we can look at the objective functions. For example, for linear regression without regularization the algorithm tries to solve the following optimization problem

$$\min_\theta \sum_i ||y^{(i)} - \theta^T x^{(i)}||^2,$$

then for Bayesian linear regression (or linear regression with regularization) the optimization problem becomes

$$\min_\theta \sum_i ||y^{(i)} - \theta^T x^{(i)}||^2 + \lambda||\theta||^2,$$

where λ is a regularization parameter.

The last remark is that Bayesian linear regression can be considered as embedded feature selection method where the size of weights θ defines the importance of the corresponding variables (zero weight means that feature is not important for the model).

3.1 Python implementation

In this section we show how regularization procedure affects the weights of the linear regression model. We generate data with 10 features with 6 out of 10 informative features:

```
In [6]: X,y = ds.make_regression(n_samples=100,
                                 n_features=10,
                                 n_informative=6,
                                 noise=1.0,
                                 bias=0,
                                 random_state=2016)
        y = y/10.
```

Different types of regularizations are implemented in the `scikit-learn` package. We will use `Lasso` class that implements L1 regularization.

```
In [7]: from sklearn.linear_model import Lasso
```

Now let us observe the model coefficients with respect to different values of the regularization term (alpha).

```
In [8]: for a in [0.0, 0.1, 0.5, 1.0, 10.0]:
            print 'alpha = ' + str(a) + ':'
            model = Lasso(alpha=a, normalize=False, max_iter=1000000)
            model.fit(X,y)
            print model.coef_
            print '***************'
```

```
alpha = 0.0:
[ -5.52107945e-03  -3.67860089e-03   7.64629577e+00   1.73280696e-01
   2.82826703e-01   2.28833747e+00   5.71482110e-03   4.49741291e+00
   5.56384364e+00   1.73914964e-03]
***************
alpha = 0.1:
[ 0.          0.          7.51109466  0.09366317  0.14180965  2.18308828
 -0.          4.39417767  5.4250924  -0.        ]
***************
alpha = 0.5:
[ 0.          0.          7.00822133  0.          0.          1.77766907
 -0.          3.9454249   4.87546516 -0.        ]
***************
alpha = 1.0:
[ 0.          0.          6.39221165  0.          0.          1.27978247
 -0.          3.36381678  4.19028074 -0.        ]
***************
alpha = 10.0:
[ 0.  0.  0.  0. -0.  0.  0.  0.  0. -0.]
***************
```

4 Online learning

The process of online learning can be described by the following diagram

$$x^{(1)} \to \hat{y}^{(1)} \to y^{(1)} \to x^{(2)} \to \hat{y}^{(2)} \to y^{(2)} \to x^{(3)} \to \dots$$

Total online error is defined as

$$\sum_{i=1}^{m} \mathbb{1}\{\hat{y}^{(i)} \neq y^{(i)}\}.$$

Examples.

- Logistic regression with stochastic gradient descent.

  1. Initialize $\theta = 0$.
  2. After i-th example, update the parameters

  $$\theta := \theta + \alpha(y^{(i)} - h_\theta(x^{(i)})) \cdot x^{(i)}$$

  (stochastic gradient descent step).

- FTRL (Follow The Regularized Leader) algorithm.

5 Advice for apply ML algorithms

The first step of designing a learning system and solving the machine learning problem is look at the data and plot it. It gives understanding about what features are given and how to work with these features.

When you solve some practical problem you can obtain different results with the same algorithm (for example, xgboost). One of the reason could be the randomness of the algorithm. To reproduce the results of the algorithm usually we fix seed when we train the algorithm. Fixed seed can help to generate constant sequence of pseudorandom numbers.

In this section we try to understand how to debug the learning algorithms. Consider some machine learning problem and choose some algorithm, as an example we choose Bayesian logistic regression implemented with the gradient descent:

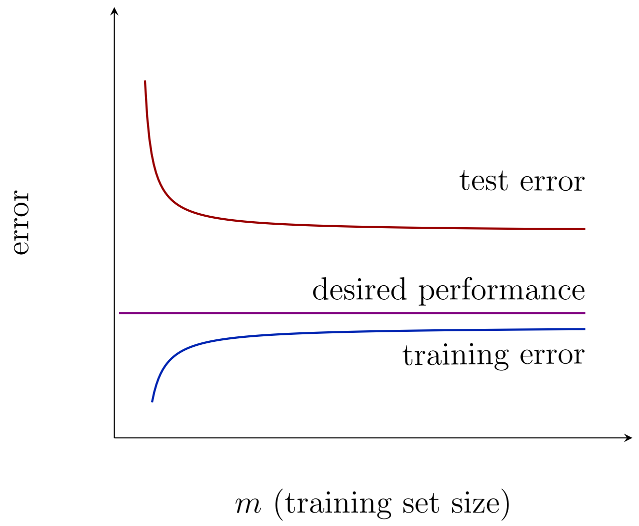
$$\max_\theta \sum_{i=1}^{m} \ln p(y^{(i)} \mid x^{(i)}, \theta) - \lambda ||\theta||^2.$$

Using cross validation procedure we get 20% test error which is unacceptably high. The main question how to reduce this error. Several possibilities are

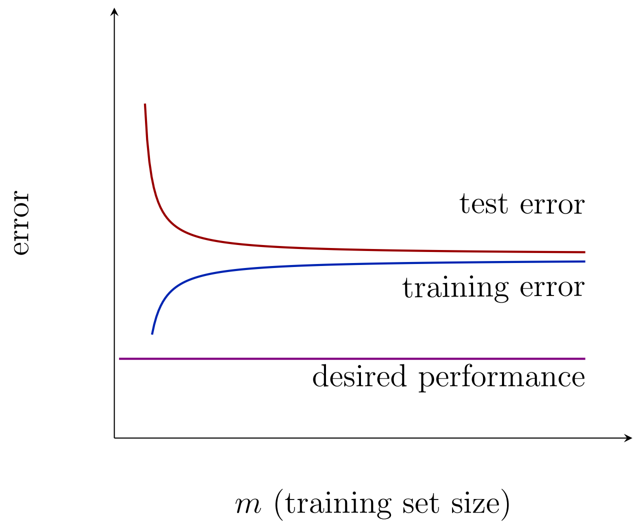
- get more training examples (fixes high variance)
- smaller set of features (fixes high variance)
- larger set of features (fixed high bias)

- change the features: for example, email header vs. email body features (fixes high bias)
- run gradient descent with more iterations (fixes optimization algorithm)
- replace gradient descent by Newton's method (fixes optimization algorithm)
- use different value for λ (fixes optimization objective)
- use different algorithm (for example, SVM) (fixes optimization objective)

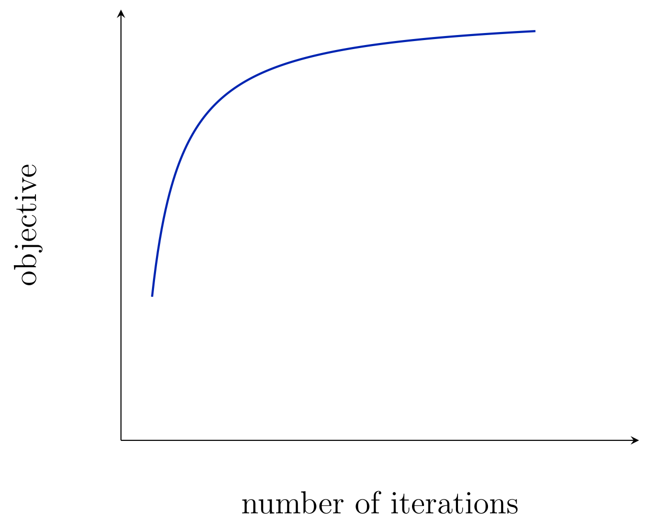
Consider some of these possibilities in more details. For example, we suspect a high variance (overfitting), then the training error will be much lower than the test error and this case is shown on the following graph (notice that gap between training and test errors is pretty big):



If we suspect a high bias (underfitting), the picture will be different:



Another diagnostics is related to the algorithm convergence. The following graph shows how the objective function changes when we increase the number of iterations:



With this graph we can understand if the algorithm converges (or if we use the correct number of iterations for the algorithm).

When optimization technique is applied we should have the correct objective function. The wrong objective function can also ruin the test error. One general technique is to add weights to the objective function

$$J(\theta) = \sum_i w^{(i)} \mathbb{1}\{h_\theta(x^{(i)}) = y^{(i)}\}$$

(for example, weights $w^{(i)}$ could be higher for non-spam than for spam).

Finally, you should carefully tune the parameters of the algorithm (for example, λ for the Bayesian logistic regression or C for the support vector machine).

In general, there are two approaches to machine learning problems solving:

- Careful design:
  - spend a long term designing exactly the right features, collecting the right dataset, and designing the right algorithmic architecture
  - implement it and hope it works

  The benefit of this approach is in the nicer and, perhaps, more scalable algorithms. We may also come up with new, elegant learning algorithms and contribute to basic research in machine learning.

- Build and fix:
  - implement something quick-and-dirty
  - run error analysis and diagnostics to see what's wrong with it and fix its errors

  The benefit of this approach is in the consumed time. Usually, it gets faster to make your application problem working, that could be important for the industrial sector.