COURSERA

Machine Learning

Contents

Skill: 3/3

Time: 3 months

Instructor: Andrew Ng (Stanford)

Contents Map

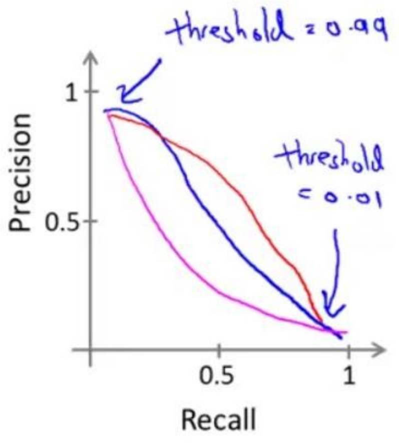
Some of the contents of the course were already summarized in other frameworks and can be found in the corresponding summaries:

- **Introduction**: standard partition to supervised (regression/classification) / unsupervised (clustering/coordinates transformation) (see Udacity AI course)
- ***Linear Regression*** (see Udacity Supervised Learning course)
- ***Logistic Regression*** (see Udacity Supervised Learning course)
- ***Regularization*** (see Udacity AI course)
- ***Neural Networks*** representation & learning (see Udacity Supervised Learning course)
- ***Unsupervised Learning*** – K-means & PCA (see Udacity Unsupervised Learning course)

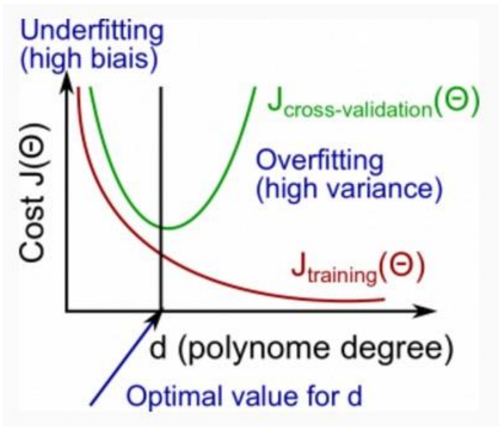
Advice for Applying Machine Learning

Learning improving can be done through getting more data samples, add/remove features, change the regularization λ and more. It is important to understand how each one of those hyper-parameters affect the learning, and how to choose what to do accordingly.

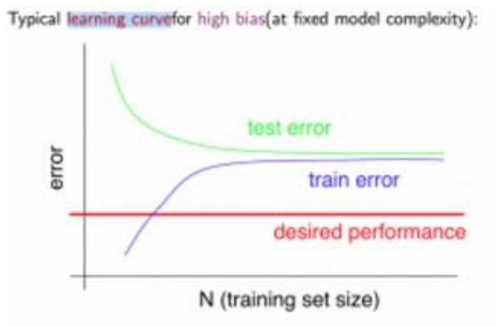
- The performance of a learning model ("correctness of the hypothesis") can be measured with relation to test data, in order to test the ***generalization*** capability.
    - Different types than total accuracy are necessary to measure performance wrt ***skewed classes*** (i.e. non-balanced classes, e.g. email classifications where 99% of the samples are non-spam).
    - For various error types and **measures of performance**, see MIT Probability & Statistics course and NLTK Book Summary).
    - In classification problems, the tradeoff between ***precision*** and ***recall*** (Pd) can be controlled through the ***decision threshold***. The threshold may be chosen to maximize the ***F-score*** (harmonic mean of the two).

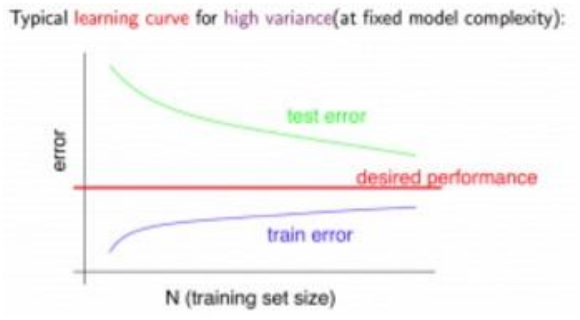


- To examine the effect of hyper-parameters, the generalization can be evaluated through ***validation data*** which is split from the training data (test data must not be used until the final evaluation).
- The error wrt the test/validation data should be calculated without the regularization term (which is used for training, not for evaluation).
- Hyper-parameters to examine wrt validation data:
    - Various **sets of features**: either essentially-different features or ***polynomial features*** (which are just different powers of existing features)
        - In this case the training error is important as well: high training error means that the problem is not poor generalization (*overfitting*), but rather poor fit in the first place (*underfitting*) due to too simple model.



- Regularization (try variant λ)
  - Note: strong regularization increases the bias (enforces simpler models)
- **Size of training data** (try to train on parts of the data with different sizes)
  - Amount of data may be much more significant than algorithm's quality.
  - Adding data reduces overfitting (more difficult to obtain dedicated fit).
  - Conditions on which more data is expected to help:
    - The features form sufficient information ("a human expert would be able to predict").
    - The hypotheses space (model architecture) is complex enough to refine the model according to the additional data.
  - ***Learning curves***: plot errors vs. different training sizes to diagnose high bias vs. high variance.





- In NN: **number of hidden layers and their sizes**

- Beginning simple, quick & dirty and then optimizing according to diagnosis (e.g. based on learning curves), is usually more efficient than trying to optimize in advance.
- Visualization of 2D surfaces in MATLAB can be done using *contour*(), which plots the isolines of a matrix.

__Summary__:

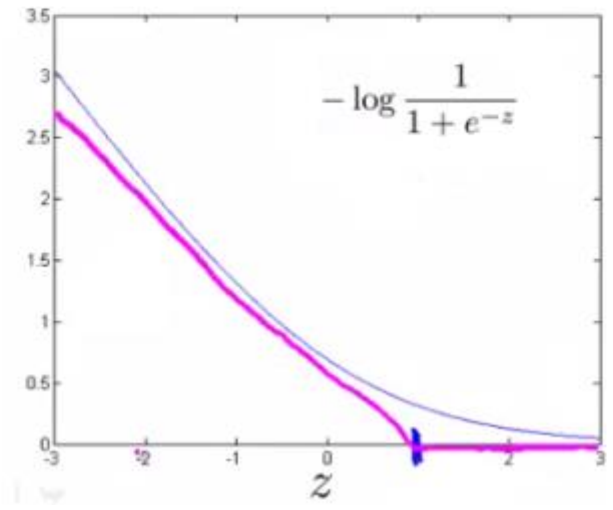
- High ***bias*** = too simple model → underfitting → large training error
- High ***variance*** = too complex model → overfitting → poor generalization → large validation error

- Generalization performance can be measured on validation set, using ***accuracy*** (for regression or balanced classification) or ***F-score*** (for skewed classification data).
- Diagnosis can be done through ***learning curves***.

Hyper-Parameter		Effect
Data	Number of data samples	Improves generalization (conditioned on the features containing sufficient info & the architecture complex enough to exploit it)
Features	Polynomial/new features	Increases variance
Hypotheses space	Model architecture (e.g. layers' number & size in NN)	Increases variance
Regularization	λ	Increases bias

Support Vector Machine

- In logistic regression, a good classification would satisfy $z \coloneqq \theta^T x \gg 0$ if y=1 and $z \ll 0$ otherwise, and the cost function is $J(\theta) = -ylog\left(\frac{1}{1+e^{-z}}\right) - (1-y)log\left(1 - \frac{1}{1+e^{-z}}\right)$.
- Another cost function is the ***Hinge Loss***, which gives larger weight to the ***margin*** around the decision boundary: correctness doesn't get extra credit for $z > 1$, and the cost of mistakes increases linearly rather than faster. For example, for $y = 1$ (i.e. desired $z \to \infty$), the cost is $cost_1(z) = \max\big(0, k(1-z)\big)$:



- $cost_0 = \max\big(0, k(1+z)\big)$ is symmetric wrt 0, and by convention the regularization is controlled by $C = \frac{1}{\lambda}$.
- In contrary to logistic regression, SVM's output does not have probabilistic interpretation, but only the discrimination $h_\theta(x) = 1 \; if \; \theta^T x \geq 0$.
  - Though in multi-class $\theta^T x$ itself is needed to choose the maximum among classes.
- **How does SVM *maximize* the margin rather than trying to achieve constant margin 1?**

- [Udacity, Supervised Learning] claims that it turns out that for $\min|\theta^T x| = 1$, the margin equals $\frac{1}{2||\theta||}$, thus the regularization probably guarantees the maximization of the margin (note that in SVM the regularization is always $\frac{1}{C} > 0$).

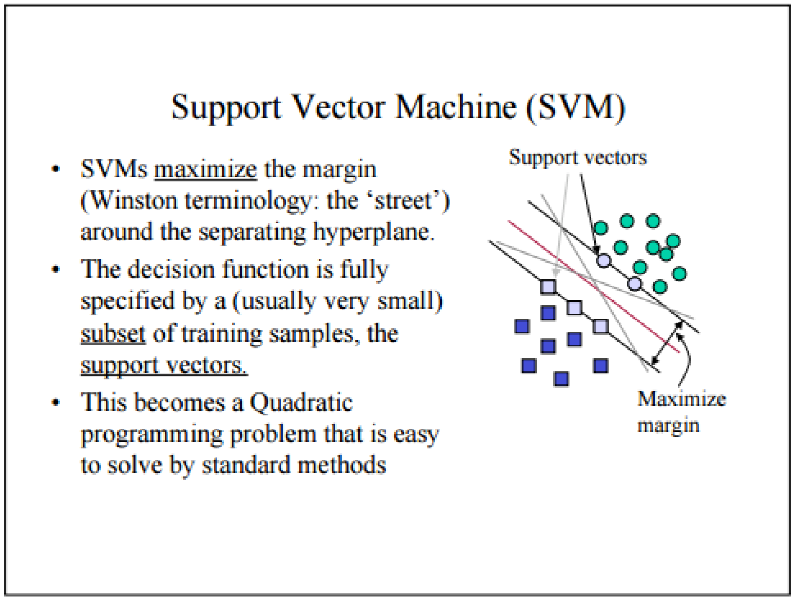
Kernels

- ***Kernels*** allow us to make complex, non-linear classifiers through generation of new features. While they are not exclusive to SVMs, the combination of the two turns out to be computationally efficient somehow.
- A kernel K measures similarity, and new features are formed based on similarity to certain landmarks $\{l_i \in R^n\}$ (i.e. $f_i \coloneqq K(x, l_i)$).
    - The landmarks can be chosen, for example, to be all the training examples (seriously?).
- For the SVM optimization not to diverge, the Kernel must satisfy ***Mencer's Condition*** (was not explained).
- ***Gaussian Kernel***: $K(x,y) = e^{-||x-y||^2/2\sigma^2})$
    - σ determines the "smoothness" of the kernel, and how to choose it was not discussed. In general, smaller σ derives sharper changes around the landmarks, which may be kind of overfitting if the landmarks are the data examples.
    - The original features of the input data should be normalized before using Gaussian Kernel.
- Linear kernel = no kernel → standard perceptron (=logistic regression).

Practical Guide

- MATLAB libraries for SVM: *liblinear*, *libsvm*.
- Hyper-parameters to choose:
    - Inverse regularization C
    - The Kernel function and its parameters
    - Landmarks to apply the Kernel on
- More complex models are recommended as m is larger than n (i.e. more samples than features, i.e. more constraints than degrees of freedom in the input). For example:
    - $n > m$ – many features, just use logistic regression.
    - $n < m$ – not many features, use SVM with a Gaussian Kernel.
    - $n \ll m$ – too few features to exploit the data – be creative and add features manually.
    - NN is claimed to work in all those cases, but be slower to train.

See more about SVMs here, e.g.:



Support Vector Machine (SVM)

- SVMs <u>maximize</u> the margin (Winston terminology: the 'street') around the separating hyperplane.
- The decision function is fully specified by a (usually very small) <u>subset</u> of training samples, the <u>support vectors.</u>
- This becomes a Quadratic programming problem that is easy to solve by standard methods

Support vectors
Maximize margin

- In the linearly-separable case, Lagrange Multipliers method can be used for analytical optimization.

Anomaly Detection

- **Learn the distribution of "normal" data, and for every new data sample x, determine the probability $p(x)$ to have such sample in the learnt distribution. Choose threshold ϵ such that $p(x) < \epsilon$ is declared anomaly.**
- Possible probability models:
    - A simple model for $p(x)$ can be ***Gaussians product***: $p(x) = \Pi p_N(x_i; \mu_i, \sigma_i)$. Learning the best parameters $\{\mu_i, \sigma_i\}$ from the training data is straight forward.
    - A bit more robust model is ***Multivariate Gaussian***, whose covariance matrix may be non-diagonal, hence it can insert implicit linear base-transformation wrt mere Gaussians product. Its price is many more degrees of freedom, used to capture correlations between the features.
- <u>Useful (instead of supervised methods) when</u> "positive" samples are easier to detect as "non-negative" rather than by common properties. For example, the following cases are appropriate:
    - Most of the data is "negative" (y=0).
    - The "positive" (y=1) data is very heterogenous.
- Common use: Fraud detection.
- Possible development method – given data of mostly non-anomalous samples:
    - Train (i.e. find p(x)) on training set of non-anomalous samples.
    - Validate and choose ϵ on validation set of mixed samples.
    - Test on mixed samples, using appropriate measure of performance (see NLTK Book summary).

- When the performance is not good enough, it usually means that the distribution of the selected features is similar for anomalous samples, hence the anomalous data should be explored for anomalous features, i.e. features that may have very unusual values in anomalies.

Recommender Systems

1. Used for recommendations of products (e.g. movies) to customers.
2. ***Content-based recommendations*** (supervised): <u>learn the preferences</u> (θ) of each customer individually, based on movies he's already rated (y), and using input features (x) such as genres.
    a. Requires features-set which is filled with values for every new sample.
    b. Requires training data from every user independently – unavailable for new ones.
    c. Requires the users to rate both positive and negative examples.
3. ***Collaborative filtering***: <u>learn the features</u> (x) from the known users rates (y) and their preferences (θ). The preferences can be asked from the users (e.g. "how much do you like action movies?").
4. More practical approach of collaborative filtering <u>learns both the features (x) and the preferences (θ)</u> using the known rates (y) and the (regularized) cost function:

$$J(x,\theta) = \sum_{\substack{m,u \\ m\ is\ rated\ by\ u}} \left(\theta_u^T x_m - y_{u,m}\right)^2 + \lambda \sum_{m,f} x_{m,f}^2 + \lambda \sum_{u,f} \theta_{u,f}^2$$

   (where the indices represent **m**ovie, **u**ser and **f**eature)
5. Note: although the problem is introduced in terms of supervised learning (x, θ and y), it **differs from classical supervised problems**: on one hand we don't have well-defined features, and on the other hand we have many parallel models that can learn from each other simultaneously:
    a. There are n_{users} separated models rather than 1 model – each model $\theta_u(f)$ has to predict its own ratings $y_u(m)$.
    b. Each model has different available training data (since each user rated different movies). Up to here it's supervised-like – one can train each model separately.
    c. However – the data samples do not have natural input features x. The relevant information for the regression (the content of the movie) is practically unavailable for the training. Instead, one must exploit the partially-filled matrix-structure of the external ratings ($y_{u,m}$) to indirectly-deduce the information of the movies.
6. Implementation note – default rates for new users:
    a. A new user u has no rate $y_{u,m}$ for any movie m.
    b. No errors cost + regularization → $\theta_u \equiv 0$ is assigned → $y_u \equiv 0$ is predicted.
    c. We would like the default preferences to correspond to the "standard" user rather than all-negative user.
    d. This is achieved by shifting the rates wrt the average rates instead of the arbitrary 0 rate:

$$\mu_m = \frac{\sum_{m \text{ is rated by } u}^{m,u} y_{u,m}}{\sum_{m \text{ is rated by } u}^{m,u} 1} = \text{average rate of movie } m$$

$$\tilde{y}_u = y_u - \mu = \text{normalized rates of user } u$$

And adjusting the linear regression to $\theta_u^T x_m + \mu_m$.

7. Note:
    a. For recommendations of the type "**people also liked…**", one can use the **similarity between movies**, derived from the learned features of the movies by $d(m_1, m_2) = \left|\left|x_{m_1} - x_{m_2}\right|\right|$.
    b. For recommendations of the type "**people also saw…**" – in which there's no rating system but only [$y_{u,m}$ = whether u visited the page of m] – one can use the same learning method. The data $\{y_{u,m}\}$ is complete in this case (for every user and every movie, we already know whether u visited the page of m or not). Indeed, the learning is not intended to predict who visited which page, but only to find pages which are similar in terms of visitors.

Large Scale Machine Learning

1. ML is mainly useful for learning complex tasks (many features) form large dbs, and nowadays 100M samples dbs are already quite common.
2. ***Stochastic Gradient Descent*** (***SGD***): repeating iterations of GD wrt the whole training data are expensive. Instead, one can apply each iteration on a single training sample, allowing many more iterations where each one is "noisier". The stochastic behavior of the iterations may cause inaccuracies, but also has the advantage of escaping local minima.
    a. Since there are many more iterations, and they're more unstable, it is reasonable to use smaller learning rate in SGD. In order to encourage convergence without being too sensitive to the random initial parameters, it is also possible to use **slowly-decreasing learning rate**.
    b. To measure the performance on the fly, it is possible to calculate the average cost frequently (say, every 1000 iterations or so).
3. ***Mini-Batch Gradient Descent***: same as SGD but with several (typically 10-100) samples per iteration. Empirically, it turns out to perform very well.
4. ***Online learning***: the SGD inherently permits us to add training data as the learning goes on, e.g. while keeping collecting data from users.
5. A learning algorithm is ***Map-Reducible*** if it can be expressed as the sum of learning algorithms over subsets of the training data. This allows parallelization of the data and the computations.
    a. All the GD variants demonstrated in the course used a cost function which was a simple sum over data samples, and the gradient is linear, thus all those algorithms are map-reducible.
        i. Though the parallelization is not cost-effective for too small mini-batches.

Application Example: Photo OCR

1. ***Object Character Recognition*** (***OCR***) problem: find text in photos and convert it to text.
2. Machine Learning ***Pipeline***: problem divided to a sequence of several modules, some of them implemented using ML.
3. ***Ceiling Analysis***: find the potential benefit from working on a certain module in the pipeline.
    a. The analysis is done by testing the change in the total performance when a specific module is manually set to have 100% accuracy (using the known ground-truth output of the module for the available data).
    b. For analysis of a sequential pipeline, the modules are artificially "fixed" cumulatively rather than independently. That is, we test the pipeline as is, then fix module 1, then fix both modules 1 & 2, then fix modules 1…3, etc. This way we see how the improvement from the current performance up to 100% is divided between the modules.
4. Photo OCR pipeline:
    a. Text detection: find rectangles containing text in the photo.
    b. Character segmentation: separate each rectangle of text to small rectangles of single character.
    c. Character recognition: identify the character correctly.
5. Text detection:
    a. Choose rectangle size, and train to classify whether a rectangle contains some text or not. Below are some positive (y=1) examples.

    b. Scan a whole photo with rectangles and detect where there's text. Use rectangles of few different sizes (stretch/compress them to correspond to the trained classifier input size), and scan with typical step size of several pixels.
    c. The typical scanning rectangles are small and local, whereas the typical text blocks are long and contain sequence of characters or even words. Thus, for every detected rectangle, we look for near detected rectangles and connect them along with the gap between them (if exists), generating long rectangular blocks of text.
6. Character segmentation:
    a. Train to classify whether a rectangle of text (with typical length of 1-1.5 character) contains a transition between two characters or not. Below are some examples.

    b. Scan every text block with rectangles, and separate to different characters wherever a transition is detected.
7. Character recognition:

   a. Just train to classify images of written characters (demonstrated before in the course).
   b. When a complete word is formed, a dictionary may be used to correct wrong classifications of characters (the probabilities assignment from the classification should be kept for this reconsidering).

Artificial Data

1. Strong ML mechanisms typically have high variance and tons of data.
2. The amount of training data can be artificially increased, generating ***synthetic data*** (or ***artificial data***), as follows:
   a. If we can **generate noiseless signal and combine it with typical noise**, it allows us to generate many different samples from scratch, using various noisy backgrounds. For example, if we want to learn to classify characters, then we can download various available fonts and past each one on various visual backgrounds.
   b. **Data samples can be manipulated** in ways which we desire that won't affect the classification. The manipulation should correspond to the expected typical distortions in the test data (adding random noise is usually not very helpful). For example, photos can be shifted, rotated, distorted, brightened etc.
3. Note: high model's variance is essential to exploit the expended data (bias/variance tradeoff can be diagnosed using the learning curves, see above).
4. Naturally, the cost of collecting more data should be taken into account when deciding whether to go on synthesizing artificial data. It is claimed that often it would turn out to be quite easy to get 10 times more data than one currently has.
   a. ***Crowd source***: services of hiring people through the web to label data, e.g. Amazon Mechanical Turk.