Studying *Machine Learning*

Qianqian Shan

Contents

Preface

The following contents contain

1. Study notes of Qianqian Shan for book *Hands-on Machine Learning with Scikit-Learn and Tensorflow*. To learn more about the book, please see http://shop.oreilly.com/product.

1 The Machine Learning (ML) Landscape

1.1 Introduction

Machine learning is the filed of study that gives computers the ability to learn without being explicitly programmed. — Arthur Samuel, 1959

Reasons for using ML:

- The programming could be shorter, easier to maintain and most likely more accurate. For example, the build of a spam filter. Save time of hand-tuning or long lists of rules.
- There are problems that either too complex for traditional approaches (for example, speech recognition) or have no known algorithm.
- ML can adapt to new data (fluctuating environments).
- $\cdots$

1.2 Types of ML Systmes

- Based on whether or not there are human supervision:
  1. Supervised learning: each training data point has a label/solution (y). Typical learning task includes classification and regression.
  2. Unsupervised learning: training data are unlabeled. Learning algorithms include
     (a) Clustering (k-means, hierarchical clustering, EM algorithm)
     (b) *Visualization and dimension reduction* (principal component analysis (PCA), kernel PCA, locally-linear embedding(LLE)
     (c) Association rule learning (apriori, eclat)
     (d) Anomaly detection: for example, detect unusual transactions, defects of products, outliers of a data set and so on.

3. Semi-supervised: training data are partially labeled. For example, for photo-hosting services, there is only one label per person but may have multiple photos per person that could be clustered (unsupervised), one can label each person in the photos (supervised, one photo may have multiple persons) for the purpose of searching photos. Most semi-supervised learning algorithms are combinations of unsupervised and supervised algorithms.
4. Reinforcement Learning: very **different** with other types. The learning system (agent) can observe the environment and perform actions, and get *rewards* in turn (or *penalties* for negative rewards). It learns by itself for best strategy (policy) to get the most reward over time. For example, Deepmind's AlphaGo.

- Based on whether or not ML systems can learn incrementally on the fly:

  1. Online: train the system incrementally by *feeding it data sequentially*, either individually or by small groups called *mini-batch*.
     (a) Each step is fast and cheap
     (b) Great for systems that receive data as a continuous flow and need to adapt to changes rapidly or autonomously.
     (c) Great when there is limited computing power/huge datasets
     (d) One concern is: how fast the algorithm should adapt to changing data (learning rate). If too fast, it will quickly forget old data; if too slow, it will have more inertia.
     (e) Cons: If bad data were fed to system, the performance will decline.
  2. Batch learning : also called offline learning. The system is trained with only available data and launched without learning anymore.

- Based on whether or not ML systems work by simply comparing with known data or detect patterns by training data with models

  1. Instance-based: the system learns the examples by heart (no model), and generalizes to new cases with a *similarity measure.*
  2. Model-based: build a model of these examples, then use that to make *predictions.*

1.3 Main Challenges of ML

1. Bad algorithm
2. Bad data

   - Insufficient quantity of training data, especially for more complex problems such as image or speech recognition.
   - Training data are not representative (don't generalize well).

- Poor quality data (errors, outliers, noise due to poor quality measurements and so on).
- Irrelevant features. The success of ML project is related to feature engineering:
  - Feature selection
  - Feature extraction: combine existing features to produce more useful ones
  - Create new features by gathering new data
- Overfitting of data: model performs well on training data but does not generalize well. It usually happens when the model is too complex relative to the amount of noise of the training data. Possible solutions:
  - Simplify the model by selecting fewer parameters / reducing number of attributes in the training data / constraining the model (regularization).
  - Gather more training data.
  - Reduce noise in the training data (fix data errors and remove outliers).
- Underfitting the training data.
  - Select a more powerful model.
  - Feed better features to the learning algorithm (feature engineering).
  - Reduce the constrains on the model (e.g., reduce the regularization hyperparameter).

1.4 Testing and Validation

1. *Testing*: split data into training set and test set and use test set to check how well the model and hyperparameters perform (generalization error, out-of-sample error).

2. *Validation*: To make full use of the data, one can use *cross validation* or *bootstrap* (sample with replacement, the not selected data can be used as validation set) for model and hyperparameters selection.

1.5 Frequentist vs Bayesian

Reference is here. Frequentist:

1. Probability is the long-run expected frequency of occurrence.

$$\Pr(A) = \frac{n}{N} \tag{1.1}$$

2. Assume that the probability parameter to be estimated is fixed (no probability distribution), while the data is random with a distribution, and the uncertainty on the parameter is caused by the limited times of data observations.

3. When data is given, can estimate the parameters with maximum likelihood estimation. (given the fixed parameter, maximize the probability to have the current data).

4. In machine learning, frequentist statistics correspond to statistical learning, find the optimized model by minimizing the empirical risk (e.g., misclassification rate, rmse).

Bayesian:

1. Probability is related to degree of belief. It is a measure of the plausibility of an event given incomplete knowledge.
2. Assume that the parameters to be estimated are random (with a prior distribution), and the data are real and fixed.
3. With given data, maximize the posterior distribution of parameters to obtain the distribution of parameters.
4. In machine learning, Bayesian statistics correspond to probabilistic graph models.
5. Applications of Bayesian statistics include cold start problem, Bayesian network (e.g., TrueSkill ranking system for game players), variational autoencoder and so on.

2 End-to-End Machine Learning Project

Go through an example project.

1. Look at the big picture
   (a) Frame the problem
      i. What is the *business objective*? How does company expect to use and benefit from this model?
      ii. Think about your component within an ml pipeline (a data *pipeline* is a sequence of data processing components).
   (b) Select a performance measure:
      i. Root mean square error (RMSE) / $\mathcal{L}_2$ norm

$$RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}[h(\boldsymbol{x}_i) - y_i]^2} \tag{2.1}$$

      ii. Mean absolute error (MAE) / $\mathcal{L}_1$ norm

$$MAE(X,h) = \frac{1}{m}\sum_{i=1}^{m}|h(\boldsymbol{x}_i) - y_i| \tag{2.2}$$

iii. $\mathcal{L}_k$ norm, for a vector $\boldsymbol{v}$ of length n,

$$||\boldsymbol{v}||_k = \left[|v_1|^k + \cdots + |v_n|^k\right]^{\frac{1}{k}} \tag{2.3}$$

The higher the norm index, the more it focuses on *large values.* So RMSE is more sensitive to outliers than MAE.

(c) Check assumptions that have been made before proceeding.

2. Get the data
   Create test set before exploring any patterns on the data (possibly on the test set).
   (a) Random sampling
   (b) Stratified sampling
3. Discover and visualize the data to gain insights, see jupyter notebook of chapter 2 at my Github.
4. Prepare the data for ml algorithms (feature engineering)
   (a) Deal with missing values (drop data with NA values; drop columns with NA values; impute the missing values: K-means, linear iterpolation,...))
   (b) Convert categorical variables to numerical (code different categories with reasonable numerical values; binary coding (0-1))
   (c) Scaling
      i. Min-max scaling (normalization), shift and rescale data so they end up ranging from 0 to 1 (or other ranges).

$$x^* = \frac{x - min(x)}{range(x)} \tag{2.4}$$

      ii. Standardization: subtracts mean and divide standard deviation (so standardized values have mean zero and variance 1). Ranges may not be 0 to 1, but more robust to outliers.

$$x^* = \frac{x - \bar{x}}{sd(x)} \tag{2.5}$$

   (d) Skewness (measure of the asymmetry of the probability distribution): if the data is skewed, check the reason and take action
      i. if it's caused by outliers, depending one the reliability of the data, can remove outliers/use spatial sign ($x^*_{ij} = \frac{x_{ij}}{\sum_{j=1}^{m} x^2_{ij}}$) to normalize features for sample i.
      ii. or take log/square root/inverse transformations on the features.
5. Select a model and train it : use cross validation.
6. Fine-tune model: tune hyperparamters; use ensemble methods; analyze the best models and evaluate the system on test set.
7. Present solution
8. Launch, monitor and maintain your system

3 Classfication

3.1 Performance measures

1. Measuring *accuracy* with *cross-validation.* Accuracy is not preferred as performance measure especially when data are skewed.

2. Confusion matrix. Count the number of times misclassified, each row represents actual class and each column represents a predicted class as in Table 1.

Predicted (H) Observed (V)	Positive	Negative
Postive	TP	FN
Negative	FP	TN

Table 1: Confusion matrix.

3. Precision and recall tradeoff.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

$$precision = \frac{TP}{TP + FP},$$

precision is the accuracy of the positive predictions.

$$recall = \frac{TP}{TP + FN},$$

recall is also called sensitivity, or true positive rate (TPR): the ratio of positive instances that are correctly detected by the classifier.

   - Check the Precision (y) - Recall (x) curve to find models.
   - It's convenient to combine precision and recall into a single metric, F_1, the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}.$$

   - F_1 score favors classifiers that have similar precision and recall. But this is not always what we want.
   - Increasing precision reduces recall and vice versa.

4. ROC curve

(a) Receiver operating characteristic (ROC) curve is used with binary classifiers. It's true positive rate $TP/(TP+FN)$ (or y) vs false postive rate $FP/(TN+FP)$ (or x).

(b) One way to compare the classifiers is to measure the area under curve (AUC). A perfect classifier has ROC with AUC equals 1.

(c) If the true positive ratio in sample is known ($p = \frac{(TP)}{TP+TN+FP+FN}$), then accuracy can be represented as

$$p * TPR + (1-p) * (1-FPR), \tag{3.1}$$

we could then draw iso-accuracy lines in parallel in ROC and move it from top left to bottom right, the first intersection of these lines with ROC indicates the best model based on the current prior knowledge and accuracy.

(d) The imbalanced number of data in different classes will also affect the choice of ROC or PR curve.

> Precision is more focused in the positive class than in the negative class, it actually measures the probability of correct detection of positive values, while FPR and TPR (ROC metrics) measure the ability to distinguish between the classes.

See more at here.

(e) Rule of thumb to choose between precision/recall (PR) curve and ROC: when positive class is rare, or when you care more about false positives than the false negatives, you should prefer PR curve.

3.2 Multiclass classification

1. Some algorithms such as random forest, naive Bayes can handle multiple classes directly.

2. One vs. all strategy: train multiple binary classifiers for each class.

3. One vs. one: train a binary classifier for every pair of digits. Each classifier only needs to be trained on the part of training data set for the two classes to be distinguished.

Note: One vs. one algorithm is preferred when the size of training data set is poorly scaled. Otherwise, one vs. all is preferred.

3.3 Error analysis

Make predictions on the training set, analyze the confusion matrix.

3.4 Multi-label classification

3.5 Multi-output classification

A generalization of multilabel classification where each label can be multiclass.

4 Train Linear Regression Models

4.1 Linear Regression

See Chapter 3 of https://qianqianshan.com/post/esl for the linear regression background. The computational complexity of finding the inverse of X^TX is typically $O(n^{2.4})$ to $O(n^3)$, so solving the model parameters via solving normal equations will be computationally expensive when the data size or number of features is large. It leads to *gradient descent.*

4.2 Gradient Descent (GD)

GD is a generic optimization algorithm of finding optimal solutions to a wide range of problems. The idea is: tweak parameters iteratively in order to minimize a cost function.

$$\theta_{i+1} = \theta_i - \eta \nabla_{\theta_i} MSE(\theta) \tag{4.1}$$

1. To find a good learning rate η, use grid search. Batch gradient descent with a fixed learning rate has a convergence rate $O(\frac{1}{iterations})$.
2. One should ensure that all features **have a similar scale**, or else it will take much longer to converge.
3. *Stochastic gradient descent*: pick a random instance in the training set at every step and computes the gradients based ONLY on that single instance.
   (a) SGD is much faster.
   (b) Can be used to train huge training sets.
   (c) The cost function will bounce up and down, decreasing only on average.
   (d) Has a better chance to find the global minimum than batch GD.
   (e) Use *simulated annealing* to gradually reduce the learning rate so the algorithm can settle at the minimum.
4. Mini-batch gradient descent computes the gradients on small random sets of instances.

4.3 Model Generalization Performance Evaluation

1. Cross-validation: tell if the model is overfitting or underfitting.
2. Learning curves: plots of models' performance (eg. RMSE) on the training set and the validation set as function of the training set size.
   (a) If the curves for training and validation sets are close and fairly high, there is an indication of *underfitting.*
   (b) If the training error is much lower, and there is a gap between the curves, there is an indication of *overfitting.*

4.4 Bias/Variance Tradeoff

A model's generalization error can be expressed as the sum of:

1. Bias: due to wrong assumptions. For example, when the data are actually quadratic, we assume linear relationship.
2. Variance: due to the model's excessive sensitivity to small variations in the training data. A model with many degrees of freedom (e.g., polynomial regression model) is likely to have high variance.
3. Irreducible error: due to the noiseness of the data itself. The only way to reduce this kind of error is to clean up the data.

4.5 Regularized Linear Models

See more in Chapter 3 of https://qianqianshan.com/post/esl for regularization and logistic regression.

Except for ridge, lasso, elastic net, a different way to regularize *iterative learning algorithms* (e.g., GD) is to stop training as soon as the validation error reaches its minimum.

4.6 Logistic Regression

Generalize logistic regression to support *multiple classes* directly without training and combining multiple binary classifiers. That is, *softmax regression or multinomial logistic regression.*

Cross entropy cost function is minimized

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^i \log(\hat{p}_k^i). \tag{4.2}$$

Softmax function is used to compute the probability that one instance belongs to a specific class. It penalizes the model when it estimates a low probability for a target class.

5 Support Vector Machines (SVM)

SVM is capable of performing linear/nonlinear classification, regression, and even outlier detection. It's well suited for small- or medium-sized datasets. It's trying to optimize the decision hyperplane (or margin), not the probabilities. **Margin** and **duality** are two key concepts in SVM.

5.1 Linear SVM Classification

It's also called largest margin classification: SVM classifier fit the widest street between classes.

1. *Support vectors*: instances located on the edge of the "street" fully determines (supports) the decision boundary.
2. SVMs are sensitive to feature scales.
3. SVMs don't output probabilities for each class.

Types of linear SVM:

1. Hard margin classification: all instances are off the street (margin) and on the correct side.
   (a) works only if the data is linearly separable.
   (b) Sensitive to outliers (e.g., even one outlier on the wrong side will make it impossible to separate).
2. Soft margin classification: find a good balance between keeping the street as large as possible and *limiting the margin violations* (instances in the middle of the street or on the wrong side). Introduce slack variable ϵ_i for each instance for the conditions of the optimization problem.

5.2 Nonlinear SVM Classification

1. Create polynomial features for linearly separable classification under new feature space. Cons: can create a huge number of features with higher degree and more features.
2. Polynomial kernel: Pros: no combinatorial explosion of the number of features since you don't actually add any features.

3. Add similarity features: add features computed using similarity function that measures how much each instance resembles a particular *landmark*.

$$\phi_\gamma(x, l) = \exp\left[-\gamma \cdot ||x - l||^2\right], \tag{5.1}$$

where l is the landmark location. Cons: may be computational expensive when the training set is large and more landmarks.

4. Gaussian radial basis kernel makes it possible to obtain a similar result as if may similarity features are added, without actually having to add the features.

5.3 SVM Regression

SVMR tries to fit as many instances as possible *on the street* while limiting the margin violations (i.e., instances off the street), and the width of the street is controlled by a hyper-parameter.

5.4 Outlier detection

See $scikit-learn.org/stable/modules/outlier\_detection.html$ for details on outlier detection with scikit-learn. The idea is: the outliers/anomalies cannot form a dense cluster as available estimators assume that the outliers/anomalies are located in low density regions, is unsupervised anomaly detection.

5.5 Hinge loss

For soft margin maximization, add relaxation/slack factor ϵ_i for each i

$$\underset{w,b}{\text{argmin}}\ \frac{1}{2}||\beta||^2 + c\sum_{i=1}^{n}\epsilon_i \text{ with } y_i(w^T\Phi(x_i) + b) \geq 1 - \epsilon_i, \tag{5.2}$$

that is, $\epsilon_i \geq 1 - y_i(w^T\Phi(x_i) + b)$, which results in hinge loss

$$\max\left(0, 1 - y_i \cdot \left[w^T\Phi(x_i) + b\right]\right). \tag{5.3}$$

The correct classifications won't contribute to hinge loss.

6 Decision Trees

Decision trees are versatile and also the fundamental components of random forests.

Decision trees are NOT **scale sensitive**.

1. CART (classification and regression tree) Algorithm: first splits the training set in two subsets using a single feature k and a threshold t_k (pairs (k, t_k) are chosen so that purest subsets are produced). *CART is a greedy algorithm, which searches for optimal splits at the top level, and then repeats the process at each level.* CART often produces a reasonably good solution, but not guaranteed as optimal.

   (a) CART cost function for classification:

   $$J(k, t_k) = \frac{n_{left}}{n} G_{left} + \frac{n_{right}}{n} G_{right}, \tag{6.1}$$

   where G_i is the impurity of subsets, n_i is the number of instances in subsets, and n is the total instances.

   (b) CART cost function for regression:

   $$J(k, t_k) = \frac{n_{left}}{n} MSE_{left} + \frac{n_{right}}{n} MSE_{right}, \tag{6.2}$$

   where $MSE_{node} = \sum_{i \in node}(y_i - \widehat{y}_{node})^2$ and $\widehat{y}_{node} = \frac{1}{n_{node}} \sum_{i \in node} y_i$.

   (c) CART is a greedy algorithm, can cannot guarantee an optimal solution.

2. Complexity:

   (a) Prediction: $O(\log(n))$ to traverse the tree.

   (b) Training: With K features, it requires to compare all features on all samples at each node, that is, $O(K \times n \log n)$.

3. Impurity measure in the cost function for classification:

   (a) Gini impurity

   $$G_i = 1 - \sum_{k=1}^{K} p_{i,k}^2, \tag{6.3}$$

   where $p_{i,k}$ is the ratio of class k instances in node i.

   (b) Entropy impurity

   $$H_i = - \sum_{k=1, p_{i,k} \neq 0}^{K} p_{i,k} \log p_{i,k}. \tag{6.4}$$

   Comparison:

   - Most of the time the two measures don't make a big difference.
   - Gini impurity is slightly faster to compute.
   - When they differ, Gini tends to isolate the most frequent class in its own branch of the tree. Entropy tends tends to produces more balanced trees.

4. Regularization: decision trees make very few assumptions about the training data, and left the tree structure unconstrained, so most likely to get overfit. So it's often called as *nonparametric* model given that *the number of parameters is not determined prior to training.*

   (a) Method 1: Use the test data, plot the correct classification rate on training data and test data vs. size/depth... of the tree. Restrict maximum depth, min samples of leaf, max leaf nodes and so on.
   (b) Method 2: Pruning. First train the tree without restriction, then prune unnecessary nodes with statistical tests such as chi-square test or reduction in variance standards and so on. See details here.

5. Issues:

   (a) Sensitive to training set rotation: decision trees love orthogonal decision boundaries (splits are perpendicular to an axis). *Solution*: may use PCA to obtain a better orientation of the data.
   (b) Very sensitive to small variations in the training data, especially when a stochastic method is used to train the data. *Solution*: random forest. *Solution*: use random forests to limit this instability by averaging predictions over many trees.
   (c) Low bias but high variance.

7 Ensemble Learning and Random Forests

Intuition: Wisdom of the crowd. Even if each classifier is weak (only slightly better than random guessing), the ensemble can still be a strong learner, provided there are *a sufficient number* of weak learners and *sufficiently diverse*.

Law of large numbers: if you keep tossing a biased coin, the ratio of heads gets closer and closer to the true probability of heads.

- The ensemble methods work best when the predictors are *independent* from one another. (*diverse classifier*)
- One solution to get diverse classifiers is to train them using *very different algorithms*.
- Another solution is to use the same training algorithm but different random subsets of the training data. For example, bagging (sampling with replacement) and pasting (sampling without replacement).

Ensemble usually has a similar bias but lower variance than a single predictor trained on the original data.

7.1 Voting classifiers

1. Hard voting classifier: Create a classifier by aggregating the prediction of each classifier (e.g., logistic regression, SVM, random forest, KNN ...) and predict the class that gets the most votes.

2. Soft voting classifier: estimate class probabilities of each classifier for each class, predict by using the highest probability. (gives more weight on highly confident votes).

7.2 Bagging and Pasting

Can be trained in parallel.

Steps:

1. Training: Bagging (sampling with replacement, one training instance can be sampled more than one time for each classifier/prediction) and pasting (sampling without replacement).
2. Ensemble: aggregate the predictions of all predictors by statistical mode (for classification) or average (for regression).

Bias and variance: the bias of each predictor is high, but aggregation reduces both bias and variance. Net result is that *the ensemble has a similar bias but a lower variance than a single predictor training on the original data.*

Comparison of bagging and pasting:

1. Bagging (bootstrapping) introduces a bit more diversity in the subsets, so bagging ends up with a slightly *higher bias* than pasting.
2. But it also means that predictors are less correlated, so the ensemble variance is reduced.
3. To conclude, bagging often has better models, generally preferred.

Evaluation: use the out-of-bag (oob) training instances to evaluate the ensemble by averaging out the oob evaluations of each predictor.

7.3 Random Patches and Random Subspaces

1. Random patches: sampling both training instances and features.
2. Random subspaces: only sampling features.

7.4 Random Forests (RF)

RF is an ensemble of decision trees. RF introduces extra randomness and diversity than bagging by searching for the best feature in a random *subset of features.*
Intuition: The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated(Wiki).

Bias and variance: RF results in no higher bias with lower variance than a single decision tree, generally yielding an overall better model.

Extra tree: More randomness is added. Based on RF, use random thresholds for each feature rather than searching for it. Much faster as the time to search splitting threshold is saved.

Usage of RF:

1. Make predictions.
2. Feature importance when doing feature selection: measure how much the tree nodes that uses one specific feature could reduce impurity on (weighted by number of training samples associated with it) average.

7.5 Boosting

Idea: train predictors sequentially, each trying to correct its predecessor.

7.5.1 AdaBoost

Idea: pay a bit more attention to the training instances that the predecessor underfitted $\Rightarrow$ new predictors focus more and more on the hard cases.

For the j-th predictor,

1. Fit a base classifier (e.g., a decision tree with max_depth = 1, decision stump), initialize the equal weights $w_i = 1/n$ for each instance i when $j = 1$, or

$$w_i = \begin{cases} w_i & o.w. \\ w_i \text{exp}(\alpha_j) & \text{if } \hat{y}_i^j \neq y_i \end{cases} \tag{7.1}$$

2. Update the weighted error rate,

$$r_j = \frac{\sum_{i=1, \hat{y}_i^j \neq y_i}^{n} w_i}{\sum_{i=1}^{n} w_i}, \tag{7.2}$$

   where y_i^j is the j-th predictor's prediction for the i-th instance.

3. Update predictor j's weight

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}, \tag{7.3}$$

   where η is learning rate.

4. Stop when desired number of predictors is reached, or a perfect predictor is found.

5. Make final predictions

$$\hat{y}(x) = \underset{k}{\operatorname{argmax}} \sum_{j=1, \hat{y}_i^j = k} \alpha_j \tag{7.4}$$

Overfitting of AdaBoost: try reducing the number of estimators or more strongly regularization on base estimator.

7.5.2 Gradient Boosting

Idea: fit the new predictor to the *residual errors* made by the previous predictor.

- Robust to overfitting.
- Reduce bias.

7.6 XgBoost

Slides.

7.7 Stacking

Train a model to perform the aggregation of different predictors (use the predictions from each predictor as input).

8 Dimensionality Reduction (DR)

Often need to do feature selection (select from original features) or feature extraction (construct new features from original features) due to curse of dimensionality given **fixed** number of samples. DR can be used to

- speed up training
- data visualization

8.1 PCA (linear)

- SVD
$$X = UDV^T \tag{8.1}$$
- Reduce dimension
- Principal component ($X\boldsymbol{v}_j$ as input) regression example (image) compression
- Probabilistic principal component analysis thinks principal analysis as the linear relationship between observed Gaussian values (y) and the *latent* Gaussian variables.
- Incremental PCA: split the training set into mini-batches and feed an IPCA algorithm one mini-batch at a time (solve the problem that data size may be too big to fit in the memory).
- Randomized PCA: use a stochastic algorithm to quickly find an approximation of the first d principal components.
- Kernel PCA: perform nonlinear projections. Good at preserving clusters of instances after projection.

8.2 Manifold Learning (can be non-linear)

Definition: a d-dimensional manifold is a part of an n-dimensional space ($d < n$) that locally resembles a d-dimensional hyperplane. For example, a 2d manifold is a 2d shape that can be bent and twisted in a higher-dimensional space ($d = 2, n = 3$).

Assumption:

- The high-dim data sets lie close to a lower-dim manifold.
- The task at hand (e.g., classification and regression) will be simpler if expressed in the lower-dim space of the manifold.

One technique: locally linear embedding (LLE), a non-linear dimensionality reduction. Idea: first measure how each training instance linearly related to its closest neighbors, and then look for a low-dim representation of the training set where these local relationships are best preserved.

To summarize, the linear PCA tries to find where the differences in data are from, can remove the possible co-linearity among features; manifold learning tries to find the representation of data in a lower dimension structure.

9 Tensorflow

10 Introduction ot Artificial Neural Networks

Artificial neural networks (ANN) is at the very core of deep learning. It's versatile, powerful and scalable, making it ideal to tackle large and highly complex ML tasks such as classifying billions of images, powering speech recognition, recommending videos etc. Reasoning of why the current deep learning wave will have a more profound impact:

- There are a huge quantity of data available nowadays and ANNs often outperform other ML techniques on very large complex data sets.
- The tremendous increase in computing power since 1990s makes it possible to train large neural networks in a reasonable amount of time.
- Small tweaks of the algorithms used in 1990s turn out to have a huge positive impact.
- Theoretical limitations of ANNs are benign in practice. For example, people may think ANN training algorithms are likely to get stuck in local optima, however, it turns out to be rare in practice.

10.1 Perceptron

1. Linear threshold unit (LTU) (Fig.10-4 on page 257): Each input is associated with a weight and LTU computes a weighted sum of its inputs ($z = \boldsymbol{w}^T\boldsymbol{x}$) and applies a step function to this sum for an output,

$$\sigma(\boldsymbol{z}) = f(\boldsymbol{w}^T\boldsymbol{x}). \tag{10.1}$$

Popular activation functions for *hidden layers*:

- Logistic function,

$$\sigma(\boldsymbol{z}) = \frac{1}{1+\exp(-\boldsymbol{z})}, \tag{10.2}$$

it has well-defined nonzero derivative everywhere, allowing gradient descent to make a progress at every step.

- Hyperbolic tangent function,

  $$\tan \mathrm{h}(\boldsymbol{z}) = 2\sigma(2\boldsymbol{z}) - 1, \tag{10.3}$$

  where $\sigma(\cdot)$ is the logistic function as above. It's continuous, differentiable and outputs range in $[-1, 1]$.

- ReLU function,

  $$\max(0, \boldsymbol{z}), \tag{10.4}$$

  even it's not differentiable at 0, it is fast to compute, and it doesn't have a max output (which reduces issues such as stucking on plateaus during GD).

2. Perceptron: a single layer of LTUs. Each LTU neuron is connected to all inputs (input neurons and optional a bias neuron, $x_0 = 1$) (Fig. 10-5 on page 258).

3. Multi-layer perceptron (MLP) is composed of input layer (passthrough), one or more layers of LTUs (hidden layers) and a final layer of LTUs (output layer). It's called deep neural network (DNN) when there are two or more hidden layers.

   - Back-propagation algorithm for training MLP: first make a prediction (forward pass), measures the error, then goes through each layer to reverse to measure the error contribution from each connection (reverse pass), and finally slightly tweaks the connection weights to reduce the error (gradient descent).

10.2 Fine-Tunning Neural Network Hyperparameters

10.2.1 Number of hidden layers

Deep networks have a higher parameter efficiency than shallow ones as they can model complex functions using *exponentially* fewer neurons than shallow nets. So it can be mush **faster** to train.

The logic behind it:
Real world data is often structured in a hierachical way and DNNs automatically take the advantage of this fact: lower hidden layers model lowe-level structures, intermediate hidden layers combine these low-level structures to model intermediate-level structures, and highest hidden layers and the ouput layer combine these itermediate structures to model high-level structures.

To conclude, for complex problems, one can gradually ramp up the number of hidden layers until overfitting. Re-use of pre-trained networks that performs similar tasks could also make the training much faster.

10.2.2 Number of neurons per hidden layer

A common practice is to form a funnel, with fewer and fewer neurons at each layer.

The logic behind it:
Low-level features can coalesce into far fewer high-level features.

One can increase the number neurons gradually until the network starts overfitting. Or starts with a model with more layers and neurons than needed and use early stopping to prevent it from overfitting.