Intel AI Academy Machine Learning course - notes

Robert Tykocki-Crow

June 21, 2020

Chapter 1

Machine Learning basics

1.1 Linear regression

Basics:

- linear combination of parameters/factors, not linear function
- based on assumption that the distribution of residuals is normal
- slower than KNN algorithm because fitting involves minimizing cost function
- linear regression models require less memory than KNN because the dataset doesn't have to be stored after fitting
- prediction is faster because it doesn't involve finding nearest neighbours
- measures of error include sum of squared errors (SSE), total sum of squares (TSS) and correlation coefficient (R2) which is equal to $1 - \frac{SSE}{TSS}$

1.1.1 Minimizing the cost function - gradient descent

The parameters are found be minimizing a cost function using e.g. the gradient descent method. This method finds the optimal parameters by following the negative gradient in respect to the β_j parameters. If only one datapoint is used at a time the method is called stochastic gradient descent. It requires less computation but takes longer to converge.

Both methods can be combined to employ what is called a mini batch gradient descent. This method is typically used for neural nets, batch sizes range from 50-256 points. Tradeoff between batch size and learning rate - typically the learning rate is gradually reduced during an epoch.

1.1.2 Improving linear regression models

If data is skewed / residuals don't have a normal distribution the data has to be transformed / standarized (standard scaling, min-max scaling). Higher order features of dataset can be captured using higher order terms, log terms and variable interaction.

1.1.3 Regularization and feature selection

Under-fitting cause larger bias, whereas over-fitting causes variance which manifests itself by a steep increase in the cross-validation error e.g as a result of using a polynomial of high order (effect similar to Runge's phenomenon). In such cases, the model fluctuates between the training data points and requires regularization.

Ridge regression (L2)

- useful for mitigation of multicollinearity in linear regression e.g. when there are many parameters

The cost function J is suplemented with an addtional term which penalizes larger coefficients because of squaring - $\lambda \sum_{j=1}^{k} \beta_j^2$. λ has to be chosen empirically.

Lasso regression (L1)

- penalty selectively shrinks some coefficients
- can be used for feature selection
- slower to converge than Ridge regression

Cost function suplemented by the following term: $\lambda \sum_{j=1}^{k} |\beta_j|$. λ has to be chosen empirically.

Elastic net regression

Cost function suplemented by the following terms: $\lambda_1 \sum_{j=1}^{k} \beta_j^2 + \lambda_2 \sum_{j=1}^{k} |\beta_j|$. λ_1 and λ_2 have to be empirically determined, they are hyperparameters. A portion of the available data has to be split for validation of the parameters - hyperparameters musn't be chosen using test data.

Feature selection

- reducing the number of features prevents overfitting (similarly to regularization)
- can improve fitting time
- improves readibility

1.2 Logistic regression

Uses logistic function to determine the decision boundary in a classification problem. The logistic function is given below:

$$\gamma_b(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x + \epsilon)}} \tag{1.1}$$

The cost function for logistical regression is different than in case of linear regression due to the fact that using a linear regression cost function for logistical regression problem would result in a non-convex function.

For cases with more than 1 feature and many categories the linear is employed as a binary decision (belongs to category or not), lines are drawn for each and then logistic regression is performed for pairs.

1.2.1 Classification problem error metrics

The results given by a classification model can be assembled into a confusion matrix:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Figure 1.1: Confusion matrix

- Type I error - False Positive, Type II error - False Negative
- accuracy: $\frac{TP+TN}{TP+FN+FP+TN}$
- recall (sensitivity): $\frac{TP}{TP+FN}$
- precision: $\frac{TP}{TP+FP}$
- specifity: $\frac{TN}{FP+TN}$
- $F1 = 2\frac{precision*recall}{precision+recall}$

Using these metrics the model can be evaluated using the Receiver Operating Characteristic (ROC):

Figure 1.2: ROC

The curve is plotted by computing the ratio at different threshold levels (decision lines). A more efficient way of evaluating the precision of the model is to plot the Area Under Curve (AUC) graph.

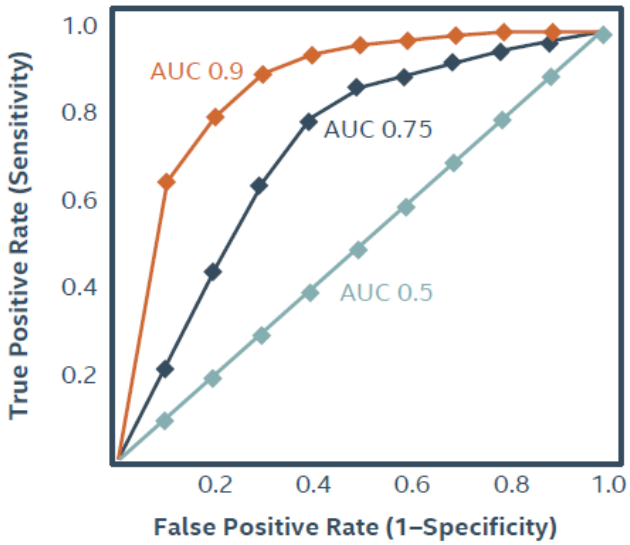


Figure 1.3: AUC

The AUC graph is based on a test of the probability that the model ranks a random positive example higher than a random negative example. One can also use the Precision Recall Curve (PR Curve) to determine the trade-off between precision and recall:

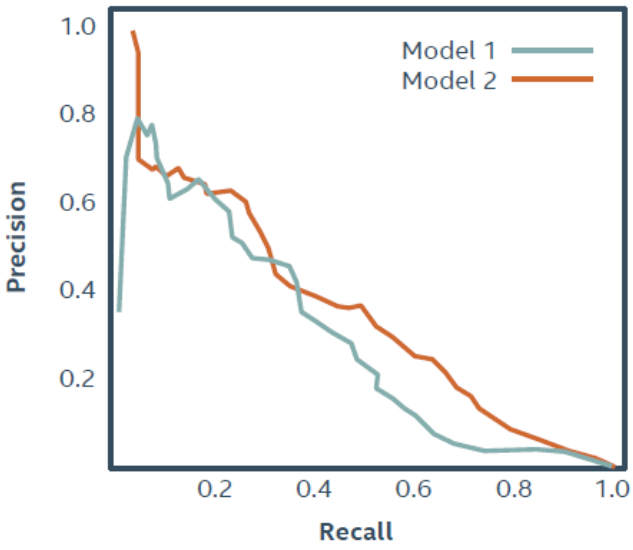


Figure 1.4: PR Curve

1.2.2 Multiple class error metrics

Most multi-class error metrics are similar to binary versions - they just require expanding as sums. E.g. accuracy for a 3-class problem is defined as:

$$Accuracy = \frac{TP1 + TP2 + TP3}{Total} \tag{1.2}$$

based on the following confusion matrix:

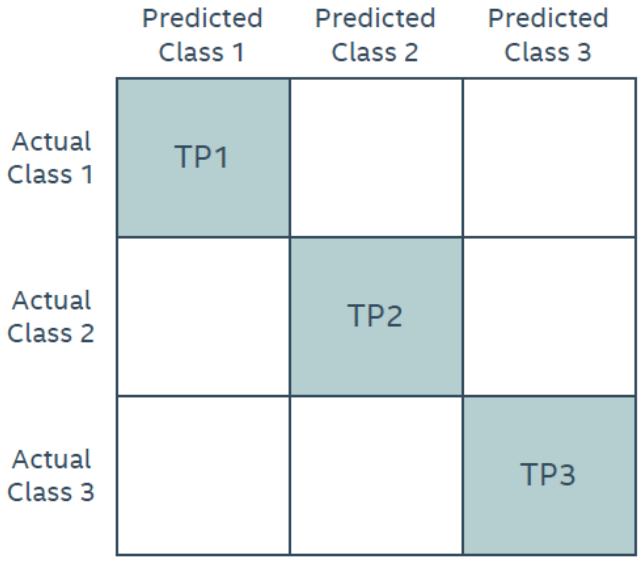


Figure 1.5: Confusion matrix for 3 class problem

1.3 Naive Bayes

$$P(Y|X) = \frac{P(X|Y) * P(Y)}{P(X)}$$

$$posterior = \frac{likelihood * prior}{evidence}$$

Figure 1.6: Bayes theorem

Naive Bayes classification neglects evidence since don't want a normalized score, only the right class:

$$P(X|Y) = P(X|Y)P(Y) \tag{1.3}$$

For a classification problem X denotes the class and Y - features. Calculating the joint probabilities by expanding therefore a naive assumption that features are independent of each other is made. Classification is performed by computing the probability for each class and choosing the class with the highest score.

1.3.1 Potential problems

To prevent underflows caused by multiplication, one might take advantage of the "log trick" - multiply logs of each element of the equation to obtain the probabilities.

Another potential problem is a zero probability of a class given a particular feature. A solution to this issue is to employ Laplace smoothing for categorical features:

$$P(Y|X) = \frac{\lambda}{N + K\lambda} \tag{1.4}$$

where N denotes the number of classes and K the number of unique options of a feature. λ is usually set to unity.

1.4 Support Vector Machines

There is a similarity between SVMs and logistical regression - their objective is to draw a decision boundary. In case of SVMs this is performed by maximizing the separation region. The interpretation of the β_i coefficients is a vector which is orthogonal to the hyperplane.

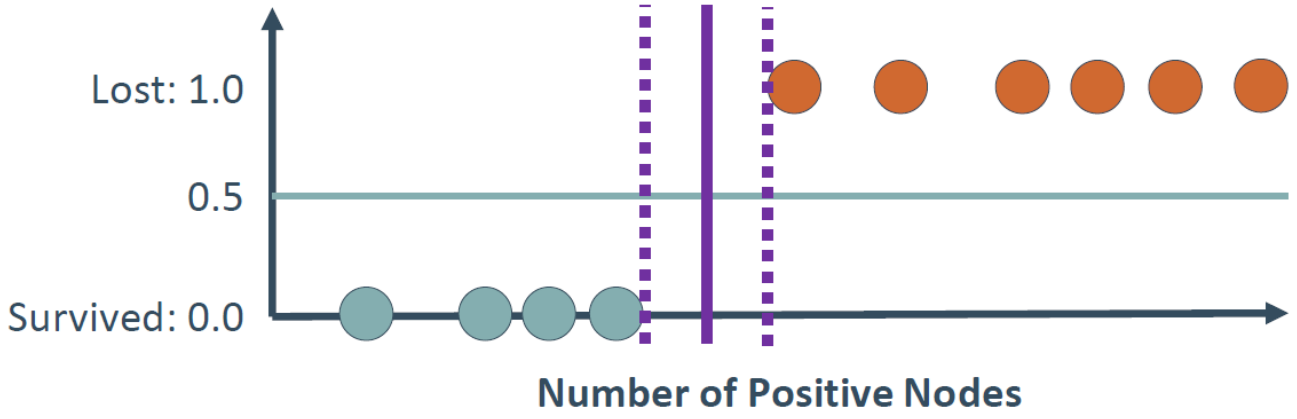


Figure 1.7: SVM - decision boundary

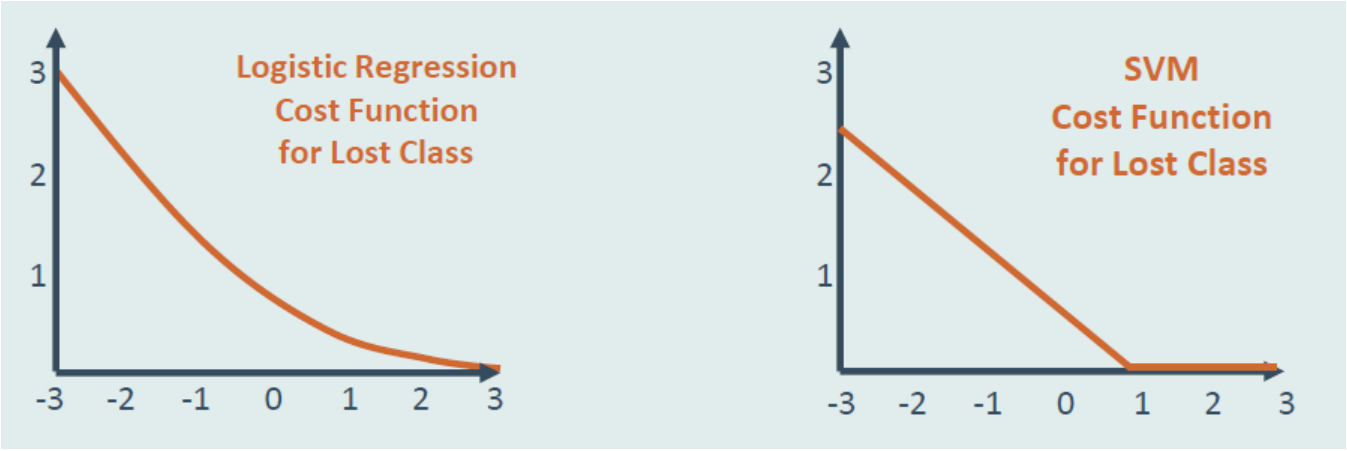


Figure 1.8: Comparison of cost functions - logistic regression and SVM

SVMs are sensitive to outliers:

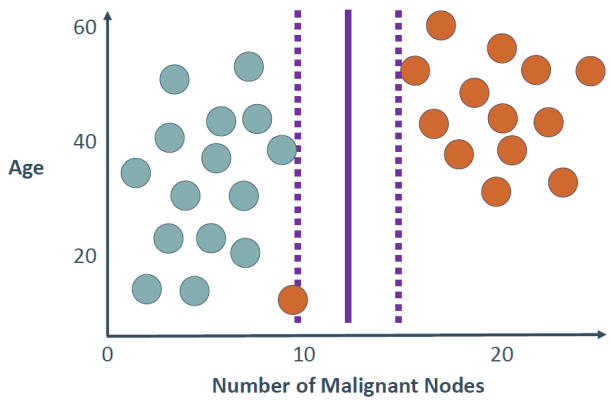


Figure 1.9: SVM - outlier sensitivity

This problem can be overcome by regularization

$$J(\beta_i) = SVMCost(\beta_i) + \frac{1}{C}\sum_i \beta_i \tag{1.5}$$

SVMs can be extended to non-linear decision boundaries by means of kernels e.g. higher order features (squares of features etc., product of two features). Another example is the SVM Gaussian kernel which applies a Gaussian function for arbitrarily chosen features (each of them). Effectively this method transforms the space to a different coordinate system. The resulting function is called Radial Basis Function.

SVMs are slow to train when many features and datapoints are analyzed. A solution to this problem is to construct an approximate kernel map with Stochastic Gradient Descent using Nystroem (subsampling) or RBF sampler and then fit a linear classifier.

1.5 Decision trees

- suitable for categorical, ordinal and continuous data (regression trees)
- features are split into binary decision - splitting ends when a) only one class remains, b) a maximum depth is reached, c) a performance metric is achieved

Greedy search - find a split which maximizes the information gained from the split. Several criteria can be chosen:

- classification error equation: $E(t) = 1 - \underset{i}{max}[p(i|t)]$. The problem with this method is that since end nodes are non-homogenous (e.g. different number of observations), no further splits would occur (total information loss using this metric)
- entropy equation solves this problem: $H(t) = -\sum_{i=1}^{n} p(i|t)log_2[p(i|t)]$
- common choice - Gini index: $G(t) = 1 - \sum_{i=1}^{n}(p(i|t)^2$

1.5.1 Pros and cons

Decision trees are easy to interpret and implement using if/else logic, handle all data categories and don't require preprocessing or scaling.

Potential problem with decision trees - overfitting. Small changes in data affect prediction greatly (high variance). The solution is to "prune" trees. Pruning can be based on the classification error threshold method.

1.6 Bagging - bootstrap aggregation

As previously mentioned, decision trees tend to overfit. A method of overcoming this issue is to create multiple trees, each using a subset of the data (approx. 2/3 of the data in one, sampling with replacement). The error of the individual tree measured on remaining data is called the "out-of-bag" error. However, fitting a bagged model doesn't produce coefficients like in e.g. logistic regression, but feature importances are estimated using oob error. Randomly permuting data for a particular feature affects the accuracy and this factor should be measured. Bagging performance increase with more trees (max improvement at approx. >50 trees).

1.6.1 Strengths

- less variability than decision trees
- trees can be grown in parallel

1.7 Random forest

Although bagging reduces variances, the subsets of data are still corellated. To further de-correlate trees one might use subsets of features - $sqrtm$ for classification problems and $\frac{m}{3}$ for regression problems. This is called a random forest.

For further randomness one can choose features and create splits randomly instead of choosing greedily - extra random trees (isolated random forest???).

1.8 Boosting and stacking

1.8.1 Boosting

The basic principle of boosting is to create consecutive classifiers by measuring residuals on the current classifier one and using them to create a decision stump on the following on.

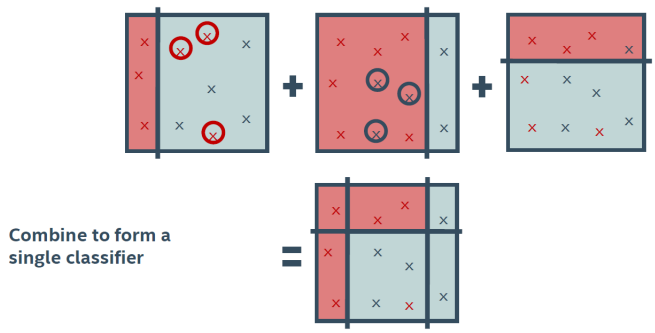


Figure 1.10: Boosting - creating consecutive classifiers

The classifiers are then combined using a learning rate λ.

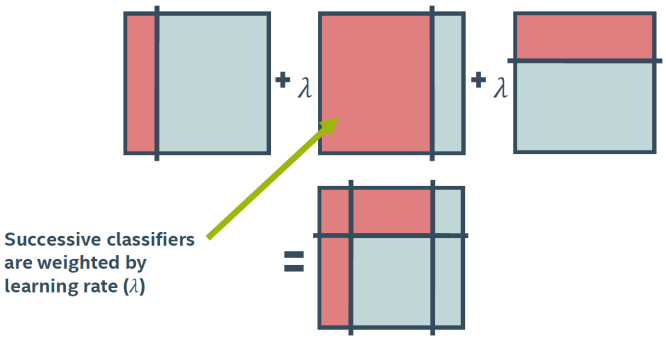


Figure 1.11: Boosting - combining classifiers

The learning rate should be < 1 to prevent overfitting (form of regularization). Boosting utilizes different loss functions. The margin is determined for each point at each stage. Value of loss function is calculated from margin.

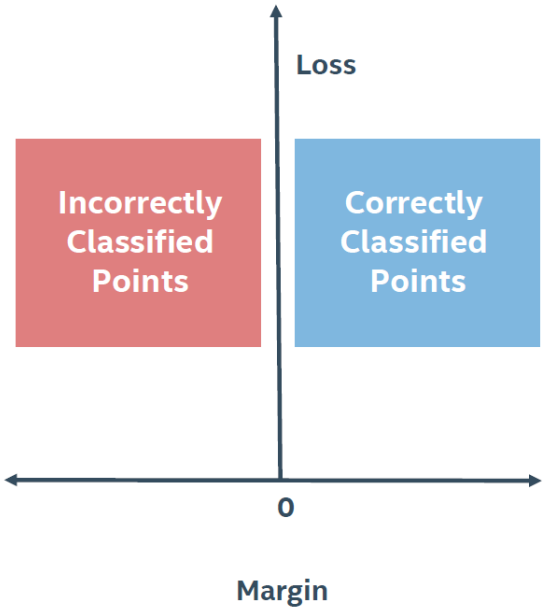


Figure 1.12: Boosting - Loss function

There are several loss functions:

- 0-1 loss function - "ideal" but difficult to optimize
- AdaBoost (Adaptive Boosting) - exponential loss function: $loss = e^{-margin}$, sensitive to outliers
- Gradient Boosting which uses different loss functions - common: binomial log likelihood loss function (deviance): $log(1 + e^{-margin})$, more robust to outliers

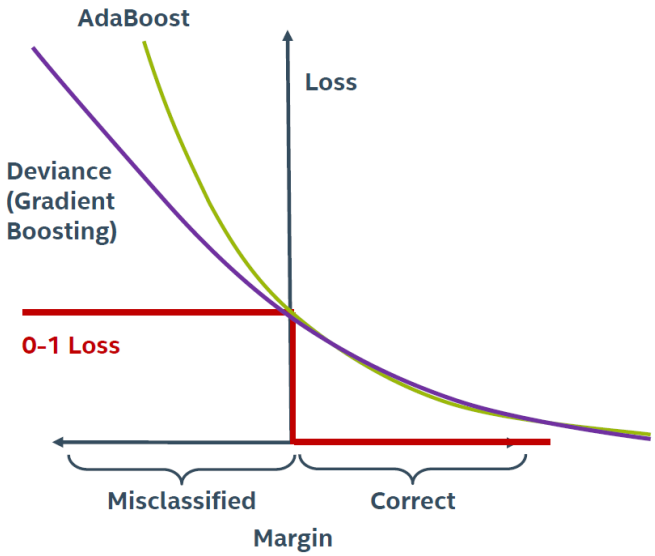


Figure 1.13: Boosting - other loss functions

Boosting is performed for entire datasets, base trees are created succesively. Problem: easily overfits (shown below), cross-validation must be used to set number of trees.

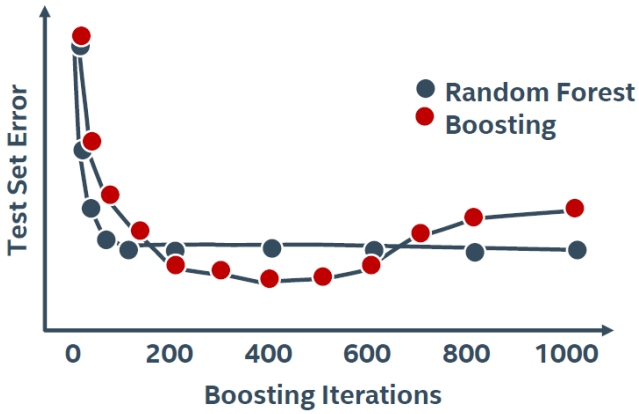


Figure 1.14: Boosting - overfitting

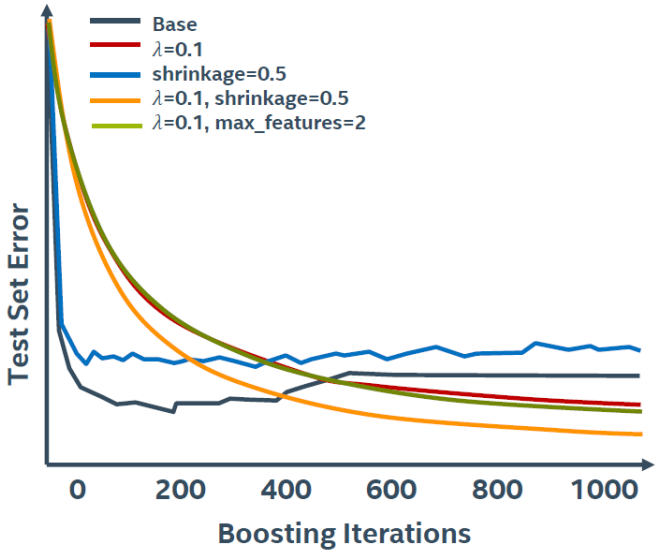


Figure 1.15: Boosting - tuning the model

1.8.2 Stacking

Stacking is combining heterogenous classifiers as shown below:

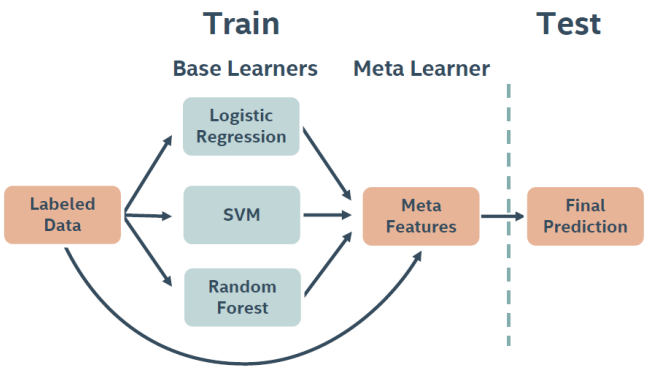


Figure 1.16: Stacking

Some data must be reserved if meta learner has hyperparameters.

1.9 Clustering

- unsupervised learning - identifying unknown structure in data
-

1.9.1 K-means algorithm

1. assign k random cluster centers
2. assign datapoints to nearest cluster center, creating clusters
3. move cluster centers to the mean value of the cluster
4. algorithm converges when points stop changing clusters

The end result depends on the initial placement of cluster centers. The metric for assesing the quality of the model is the sum of squared distances from each point to its cluster, also known as inertia. The model can be fitted several times to choose the best version based on the metric.

1.9.2 Distance metrics

- Euclidian distance (L2), standard but sensitive to the curse of dimensionality
- Manhattan distance (L1, city block distance)
- Cosine distance: $cos(\theta) = \frac{A \cdot B}{||A||||B||}$, better or text data where location of occurence is less relevant
- Jaccard distance - applies to sets (word occurence): $1 - \frac{A \cap B}{A \cup B}$

1.9.3 Hierarchical Agglomerative Clustering algorithm

1. find the pair of closest points, merge
2. find the next pair of closest points and merge
3. if closest pair is two clusters, merge them into a larger cluster
4. stop when desired number of clusters is achieved or minimum average cluster distance reaches a set value

Other linkage (stop criterium) methods are possible, e.g. complete linkage - max pairwise distance between cluster and all other points or average linkage - average pairwise distance between clusters and all other points. Merging of clusters can be performed using Ward linkage - based on best inertia.

Other types of clustering:

- mini-batch k-means
- affinity propagation
- mean shift
- spectral clustering
- ward clustering
- DBSCAN

1.10 Dimensionality reduction

The number of training examples required grows exponentially with dimensionality. Dimensionality is reduced by choosing subsets of features, omitting those which don't provide additional information - those which are correlated with other features. In example, when two features correlate we can create a new feature as a combination of both (Principal Component Analysis).

1.10.1 Single Value Decomposition

Matrix factorization method used for PCA. PCA and SVD seek to find the vectors that capture the most variance, however they are sensitive to axis scale, so data has to be scaled. Transformation performed by PCA/SVD are linear. Although the method can be used for non-linear data, dimensionality reduction can fail. Similarly as in SVM, kernels can resolve problems with non-linear data.

1.10.2 Multi-dimensional scaling

- non-linear transformation
- doesn't focus on maintaining overall variance, but geometric distance between points
- used for highf dimension data, NLP, image-based datasets

Chapter 2

Deep Learning basics

2.1 Neural Networks

- computation engine built from layers containing activation functions - each node of a layer is a function with inputs from the previous layer; layers do not have to be linearly stacked;
- each neuron is a unit of regression - larger networks are needed because one neuron only permits a linear decision boundary;
- input layer, hidden layers, output layers;
- net input - sum of weighted inputs which is fed to the activation function, activations - outputs of activation functions;
- epoch - single pass through all of the training data;
-

2.2 Training neural nets

Training is employed by computing an output, comparing the output to the correct results, computing the loss function and then apply backpropagation - adjusting weights. Calculating the gradient for neural nets is achieved using the chain rule. First, derivatives of the loss function with respect to every weight in the network are computed. The activation functions tend to have handy derivatives, e.g. the sigmoid function $f(z) = \frac{1}{1+e^{-z}}$ has the following derivative: $f'(z) = f(z)(1 - f(z))$. According to the chain rule, the loss function derivatives are computed sequentially layer by layer, starting from the last one.

$$\frac{\partial J}{\partial W^{(3)}} = (\hat{y} - y) \cdot a^{(3)}$$

$$\frac{\partial J}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot a^{(2)}$$

$$\frac{\partial J}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right) \cdot X$$

Figure 2.1: Chain rule - each term can be interpreted as an estimate of the change of activations with respect to the input, backpropagation is employed by multiplication of terms from the following layers.

The problem with the sigmoid function is that the gradients become very small for early layers ("vanishing gradient" problem).

2.2.1 Other activation functions

- hyperbolic tangent function $tanh(z) = \frac{sinh(z)}{cosh(z)} = \frac{e^{2x}-1}{e^{2x}+1}$
- rectified linear unit (RELU)

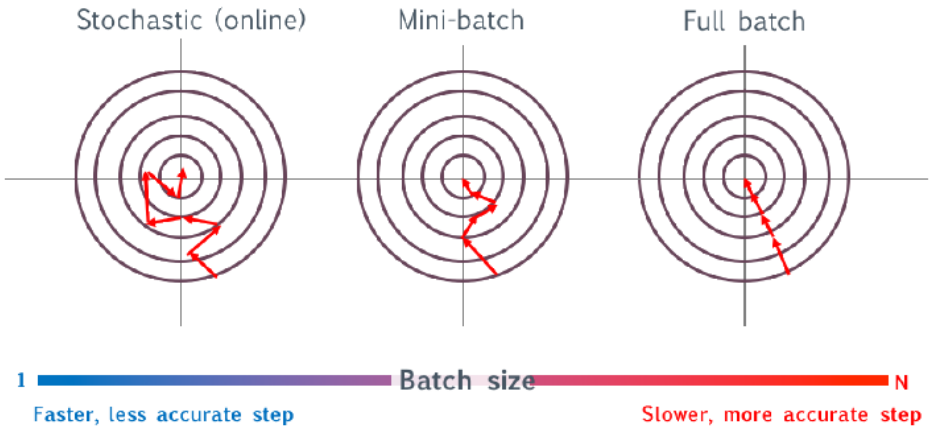
$$ReLU(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$$

- leaky rectified linear unit

$$LReLU(z) = \begin{cases} \alpha z, & z < 0 \\ z, & z \geq 0 \end{cases}$$

2.2.2 Updating weights

In the classic approach, derivates are obtained for the entire dataset, however this requires a considerable computational effort, so the algorithm becomes slow for large datasets. Stochastic gradient descent can be a remedy to this problem - the steps are less informed but their number is higher and the introduced noise should be balanced out over time. When using SGD, a smaller step size is needed, and data regularization usually helps. Alternatively, one can choose a compromise approach - mini-batch GD (batches of 16-32 data points).



In full batch GD, one step per epoch. In SGD / Online, n steps per epoch (n = training set size). In minibatch - n / batch size. A metric for learning is the number of epochs. To avoid cyclical movement effects, it is recommended to shuffle the data after each epoch - this ensures that the chosen batches are random.

The inputs should be scaled to the $[-1, 1]$ or $[0, 1]$ interval. Furthermore, one can standarize (make the variable approximately std. normal) the inputs:

$$x_i = \frac{x_i - \hat{x}}{\sigma} \tag{2.1}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x})^2} \tag{2.2}$$

2.2.3 Classification problems with neural nets

- for binary classification, the final layers has a single node and a sigmoid activation funtion; output between 0 and 1 and can be interpreted as probability; handy derivative; analogous to logistic regression;
- for multiclass classification the final layer is a vector with the length equal to the number of classes, utilizes the *softmax* sigmoid extension to multiclass problems: $softmax(z_i = \frac{e_i^z}{\sum_{k=1}^{K} e_k^Z})$. Categorical cross entropy is used as the loss function: $C.E. = -\sum_{i=1}^{n} y_i log(\hat{y}_i)$

2.2.4 Regularization techniques for deep learning

In order to regularize - prevent overfitting - the following measures can be taken:

- regularization penalty in the cost function - e.g. explicitly adding a penalty to the loss function for having high weights (similar to Ridge Regression):

$$J = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda\sum_{j=1}^{m} W_i^2 \tag{2.3}$$

- dropout - a technique where at each training iteration (batch) we randomly remove a subset of neurons; makes the network more robust because it doesn't rely on individual pathways; at test time, the weight is rescaled based on the time it was active;
- early stopping - choosing rules as to when the training is stopped, e.g. check log-loss function every 10 epochs to determine whether the model hasn't started to overfit;
- stochastic / mini batch GD also help to some degree

2.2.5 Optimizers

Optimizers are tools for determining the update of weights. The standard formula $W := W - \alpha\nabla J$ often provides unsatisfactory performance. Momentum optimizers smooths out the variation of standard updates, changing weights a little bit at every update. E.g. Nesterov momentum:

$$\nu_t = \eta\nu_{t-1} - \alpha\nabla(J - \eta\nu_{t-1}) \tag{2.4}$$

$$W := W - \eta_t \tag{2.5}$$

where η is the momentum. The idea is that overshooting is controlled by looking "ahead". A different approach si the AdaGad optimizer. The idea is to scale the update separately for each weight. Sum of previous updates is kept and new updates are divided by factor of the previous sum.

$$W := W - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \nabla J \tag{2.6}$$

A popular method nowadays is the RMSPROP which is similar to AdaGrad. However, rather than using the sum of previous gradients, older ones are decayed more than recent ones. Therefore, it is more adaptive to recent updates. Another popular method - ADAM - uses both 1st and 2nd order change information and decayover time.

2.3 Convolutional Neural Networks - CNNs

So far the structure of input data was not constrained - inputs were treated interchangeably. However, not all data-driven problems, are free of relationships between data points. Images are an example of such data - they are characterized by the following traits:

1. pixels have a topology
2. lighting and contrast
3. human visual system
4. nearby pixels tend to have similar values - the role of gradients
5. edges and shapes
6. translation and scale invariance

If images were treated in the same way as regular data, that is if each pixel had its own weights, the number of weights would be tremendous and so would be the variance. We have to introduce a bias to look for patterns.

2.3.1 In a nutshell

- built up from kernels - grids of weights overlaid on an image, centered on one pixel; used for traditional imahe processing techniques: bluring, sharpen, edhe detection, emboss;

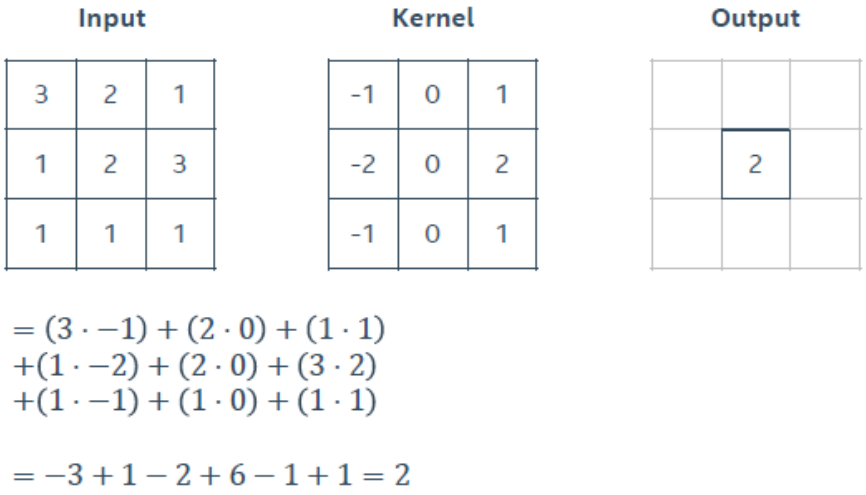


Figure 2.2: Kernel example

The neural network learns which kernels are most useful, uses them over the entire image (translation invariance). Effectively, the number of parameters is reduced whereas variance is mitigated. The dimensions of a kernel are referred to as grid size, typically it is such that a central point can be distinguished. As a consequence of this kind of discretization of an image, the network suffers from an edge effect as a center pixel cannot be determined for corners. This issue is resolved by padding - adding 0 value (typically) pixels around the frame of the image.

2.3.2 Convolution settings

- stride - the step size between neighbouring kernels, if greater than 1 than the output dimension is reduced;

- depth - number of kernels used for learning, often the number of channels describing a pixel, RGB -3, CMYK-4;

Beyond increasing the stride, another way of reducing the problem size is pooling - mapping a patch of pixels to a single value. There are several methods for choosing a numeric value to represent a patch - e.g. maximum value (max-pool), average of values (average-pool).

2.3.3 Convolution structure

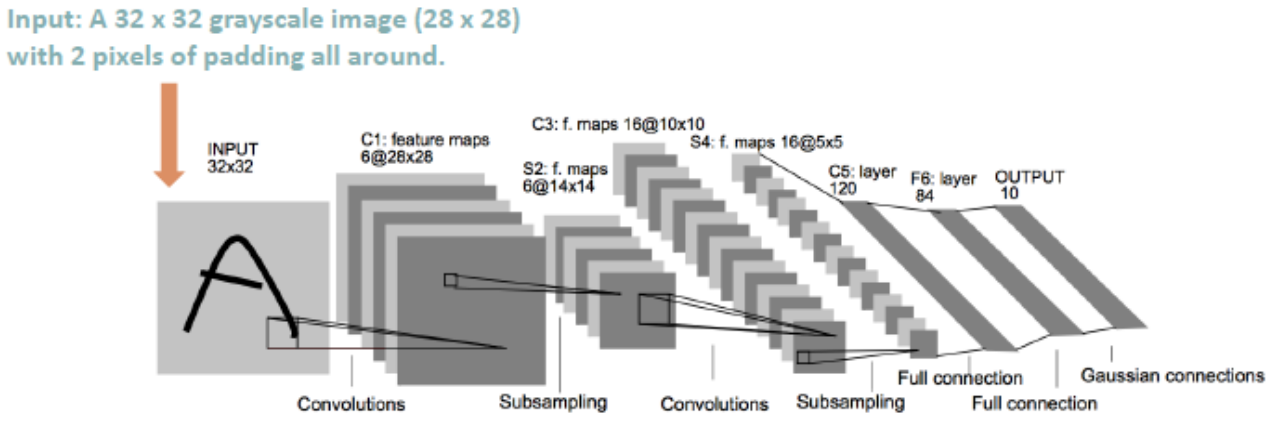


Figure 2.3: Le-Net 5 CNN architecture

1. input;
2. convolutional layer, depth=6 (six different kernels used for learning), making the output $28x28x6$;
3. pooling layer with stride 2;
4. convolutional layer on downsized data, depth is increased to 16;
5. another pooling layer;
6. then the kernels are flattened to a vector whose size is reduced by a series of fully connected layers;
7. softmax output with size equal to number of classes;

2.3.4 Training convolutional neural networks

1. the first layers are the hardest (i.e. slowest) to train due to the vanishing gradient issue; at the same time, they are responsible for capturing primitive features which are quite general, not case specific;
2. later layers capture features which are more specific to a particular classification problem, they are easier to train since adjusting their weights has a more immediate impact on the final result;

2.4 Transfer learning

There are several difficulties associated with training a model from scratch:

- very large datasets;
- many training iterations;
- computational cost;
- time for hyper-parameter optimization.

However, general features captured by early layers should generalize and the weights can be easily stored. Transfer learning is employed by using the early layers of a pre-trained network, re-training the later layers for a specific application. Additional training of a pre-trained network on a new dataset is referred to as fine-tuning.

2.4.1 Guidelines for fine-tuning

1. The higher the similarity of the data and problem definition between the new case and the original one, the less fine-tuning is necessary.
2. The more data is available for the new problem, the more benefit is taken from longer and deeper fine-tuning.
3. The new data cannot differ substantially in form with respect to the data that the model was trained on.