INTRO

Supervised Learning

The machine learning task of learning a function that maps an input to an output based on training input-output pairs.

Examples :

- Classification : aims at giving a label to every element of a set of objects.
- Regression : predict real valued labels.

Pipeline

1. Representation : standardised data representation is required, most often vectors.
2. Model Fitting : given training examples as input, try to derive a decision rule.

   Goal : fin balance between "goodness" of the fit and its complexity.
3. Prediction : apply the model to test data to label it.

Unsupervised Learning

Type of learning that helps to find previously unknown patterns in a data set without preexisting labels.The goal is often to obtain a compact representation of data.

Examples :

- Clustering : given as input a data set without labels assign clusters.
- Dimension Reduction : input is high-dimensional data, the goal is to approximate the input in much lower dimensions.

Other Specialized Learning

- Semi-supervised Learning
- Transfer Learning : learn on one domain, apply on another
- Active Learning
- Online Learning : learn from examples as they arrive over time
- Reinforcement Learning : learn by interacting with the environment."

GENERAL STUFF

Regression

- Simple Model that makes a lot of mistakes is said to have a high bias, low variance.
- Complex Model that does well on its training data is said to have low bias, high variance.
-

PART 1 - SUPERVISED LEARNING

Regression

Linear Regression

Attempts to model the relationship between two varaiables by fitting a linear equation to observed data. There is an explanatory and a dependent variable linked by the following equation : $Y = a + bX$, with b the slope of the line and a the intercept.

Goodness of the Fit

Is foud by finding the representation that gives the smallest residual : $r_i = y_i - w^T x_i$. The optimal weight vector can be found via the least-squares method, using one of two approaches :

- Closed Form : solve $w* = (X^T X)^{-1} X^T y$, with X a matrix consisting of rows of x_i
- Optimisation : by gradient descent.

High variance, low bias model

Gradient Descent

Iterative Optimisation Algorithm for finding the minimum of a function.

How it works :

1. start at an arbitrary $w_0 \in \mathbb{R}^d$
2. for $t = 1, 2...$ do the following : $w_{t+1} = w_t - \mu_t \Delta R(w_t)$ with μ_t is the learning rate.

Convergence

If the step size is small enough, the gradient descent converges to a point, and for convex functions it finds the optimal solution.

If the step size is chosen poorly we move away from the optimal solution in each step.

Ways to chose the step size adaptively :

- Line Search : optimize at every step, i.e. using the min function.
- Bold Driver Heuristic : increase step size if function decreases and decrease if it increases

Complexity

- Closed Form : computation of one equation in $O(nd^2)$, solve system in $O(d^3)$
- Gradient Descent : computes gradient in $O(nd)$, need to repeat it $O(log(\frac{1}{e}))$ times => Gradient Descent is more efficient.

Properties

- GD is sensitive to initial conditions: depending on where one starts , one will end up at a different local minimum
- GD can take exponential time to escape saddle points.
- GD treats all directions in parameter space uniformly : learning rate is the same in all directions.

Fitting non-linear Functions

???

MODEL SELECTION & GENERALIZATION

Overfitting and Underfitting

Overfitting : a model that trains the training data too well. When the the noise in the training data is picked up and learned as concepts by the model. The problem is that they do not apply to new data and negatively impact the models ability to generalize.

Underfitting : a model that can neither model the training data nor generalize to new data. Not often discussed as it is easier to detect given a good performance metric.

Generalisation

???

Learning from Finite Data

We only have a finite amount of data, hence overfitting can occur in this case.

=> the expected value of the **empirical risk** (= estimated true risk on sample data set) is ≤ the expected value of the expected error (=true risk).

Evaluation for Model Selection

We are not just interested in how good a model is but also in choosing the best amongst a set of models.

Process : pick the model that does the best on a test set.

Problem : optimizing for a single test set creates a bias -> overfitting

Solution : pick multiple test sets and avergae over them.

CROSS VALIDATION

Principle: if you choose data to make some decision on a model you cannot use the same data for evaluation purposes anymore.

Outline

```
For each model m, repeat the following k times :
  1. split the data into n different data sets
  2. train the model on the different splits
  3. estimate the error
Based on the k repetitions, select the best model (with the smallest
error on average)
```

Design Choices

How to split the Data

- **Monte Carlo:** pick validation set a t random, train on the remainding data => avergae validation error
- **k-fold:** partition data into k folds, train on k-1 and evaluate on the remaining => vary evaluation fold and average over runs.

How to pick k

- small k : this leads to smaller training sets which may lead to underestimating the performance of the model and also potentially overfitting to the test set and underfitting to the training set
- large k: better perf in general, but leads to higher complexity.

Regularization

When using non-polynomial models, we need a new measure to see when we are starting to overfit.

=> large weight/ coefficients seem to imply overfitting because weights tend to increase in more complicated models.

Lasso Regression

This model uses **Shrinkage**, which is where data values are shrunk towards a central point as the mean. this procedure encourages simple and sparse models, and is well-suited for models showing high levels of multicollinearity or when you want to automate parts of the model selection.

Easier to interpret than Ridge Regression.

Selects a subset of the input features.

L1-Regularisation

Adds a penalty that is equal to the absolute value of the magnitude of the coefficient.

Renormalizing the Data

If we penalise large weights we need to renormalize data such that it has zero-mean and unit-variance.

Ridge Regression

When LeastSquares determines values for the parameters in the equation :
$P1 = y_{intercept} + slope * P2$ it minimizes the sum of the squared residuals. Instead, when Ridge Regression determines the values of the parameters in the equation, it minimises the sum of the squared residuals + $\lambda * slope^2$.

The larger λ gets the less sensitive to p2 the predicitons for p1 become.

high bias, low variance model

CLASSIFICATION

Task of supervised learning, where we want to assign a lebel y to data points.

Linear Classifiers

Classify data into labels based on a linear combination of input features, hence they can only be used to classify data that is linearly separable.

We search : $h : \mathbb{R}^d \rightarrow \{-1, 1\}, x \rightarrow sign(w^T x)$ => find the optimal weight vector w.

Finding Linear Separators

???

Variants of Gradient Descent

Stochastic Gradient Descent

GD methods are good but they are expensive because they require summation over the entire training set. Hence the idea : evaluate the gradient only on a set of random points.

Theorem: SGD is guaranteed to converge if $\sum_t \mu_t^2 < \infty$

Perceptron Algorithm

A linear classification algorithm using SDG on the perceptron loss function l_p with learning rate $\mu = 1$. The perceptron will obtain a linear separator <=> the data is linearly separable.

Support Vector Machines

One of the most used linear classifiers, work in the same way as the perceptron, but use the **hinge loss function** defined as $l_H(w; x, y) = max(0, 1 - yw^T x)$.

Theorem: if the hinge loss is 0, the SVM will linearly separate the data. => not as strong as the perceptron as the implication is only one-sided.

FEATURE SELECTION

In high-dimensional problems we may prefer to not work with all available features (generalisation, complexity…)

Methods

- Filter Methods: identify relevant features by estimating some distributional quantity
- Wrapper methods: identify useful features based on model performance.
- Intrinsinc Methods: identify useful features jointly with model learning and construction.

Greedy Feature Selection

Add or remove features to maximise cross-validation prediction accuracy. In general, it will not find the optimal solution and it is computationally heavy and need to retrain model for every feature combination, but it is fast.

Forward

- Typically faster if fewer relevant features
- dependency handling between features is poor

Backward

- better at handling dependencies
- need to train several models for higher dimensions.

Joint Feature Selection & Training

???

NON-LINEAR PREDICTION WITH KERNELS

Perceptron / SVM revisited

Idea : reformulate the perceptron problem in $\mathbb{R}^D$ as a problem in $\mathbb{R}^n$ where $n = |D|$, which is good because often $d > n$.
=> the objective only depends on inner product of pairs of data points => allows us to work in higher dimensions as long as the inner products can be computed efficiently.

Kernels

Let ϕ be a non linear feature trasnformation such that :
$x \to \phi(x), x^T x' \to \phi(x)^T \phi(x') =: k(x, x')$, k being the kernel.
The kernel is an efficient inner product.

How to derive kernel formulation

1. Look for a way to reformulate with a linear kernel
2. Consider solutions that are some linear combination of the data
3. Reformulate such that X appears as $X^T X$
4. Replace $X^T X$ with kernel matrix

Polynomial Kernels

With d features in our feature vector x:

- $k(x, x') = (x^T x')^m$ represents all monomials of degree m,
- $k(x, x') = (1 + x^T x')^m$ represents all monomials of degree ≤ m

Kernelled Perceptron

???

Properties of Kernel Functions

Let X be the data spacem then a kernel is a function $k : X \times X \to \mathbb{R}$ where :

- k is symmetric : $k(x, x') = k(x', x)$
- k is an inner product in a suitable space

=> kernels are semi-definite matrices, and every semi-definite matrix is a kernel.

Theorem (Mercer): ???

Example Kernels in $\mathbb{R}^d$

- **Gaussian Kernel:** $k(x, x') = exp(-\frac{\|x-x'\|_2^2}{h^2})$
- **Laplacian Kernel:** $k(x, x') = exp(-\frac{\|x-x'\|_1}{h^2})$

where h is the length scale parameter.

Kernel Engineering

Kernel Properties

- Computing solutions depends on the sample size instead of the dimension of the features
- Picking or designing a kernel requires prior knowledge about the problem
- instead of designing a new kernel, one can combine nknown kernels

Theorem: Let $k_1, k_2 : X \times X \to \mathbb{R}$ be two kernels, then the following are kernels too :

- $k_1(x, x') + k_2(x, x')$
- $k_1(x, x') * k_2(x, x')$
- $k_1(x, x') * c, c \geq 0$
- $f(k_1(x, x'))$ if f is polynomial with positive coefficients or exponential function.
- $k(z, z') = k(v(z), v(z))$ with $v : Z \to X$

Nearest Neighbour Classifier

Idea: predict the majority of labels of the k nearest neighbours. the parameter k is chosen by cross-validation, with choosing it too large leads to undercutting and choosing it too small leads to overfitting.

Pro's: requires no training, just prediction.

Con's: prediction depends on all the data.

Difference to kernelized Perceptron: it has no weights, and also the kernelized perceptron is able to capture global trends and only depends on wrongly classified examples.

Parametric VS Non-Parametric Learning

- Parametric : fininte set of parameters
- Non-Parametric : grows in complecity with the size of the data, hence it is more expensive and more complex

=> kernels allow us to derive non-parametric models from parametric ones.

Kernelized SVM

???

Kernelized Linear Regression

???

Semi-parametric Models

Sometimes parametric models are too rigid while non-parametric ones fail to extrapolate, hence we need to use additive combination of linear + non-linear kernel functions. This can better model linear trends that show some periodicity within, for example.

CLASS IMBALANCE

Sometimes the data is imbalanced (e.g. there are many more positive than negative values). With that the following issues arise :

- Performance Metric : accuracy is not good anymore.
- Empirical Risk: minority class conntributes little to the risk and thus may be ignored during optimisation,

Subsampling & Upscaling

- **Subsampling:** remove training data from the majority class to obtain a balanced data set
  => faster but wastes available data
- **Upsampling:** repeat data from minority class.
  => uses all data but is slower.

Cost-Sensitive Classification Methods

Idea: Modify the perceptron or SVM to take the class imbalance into account.

Performance Metrics

1. Accuracy

=> The ratio of correctly classified data points to all data points.

2. Precision

=> the ration of correctly predicted positive observations to the total predicted positive observations.

3. Recall

=> the ration of correctly predicted positive observations to all obervations in the actual class.

4. F1-Score

=> the harmonic avergae of precision and recall

Tradeoffs

???

MULTI-CLASS PROBLEMS

Idea : instead of having only one label, binary decision, we have a set of labels.

One-VS-All Approach

Idea: solve n (=number of labels) binary classifiers, one for each class that separates class members from non-members and classify using classifier with largest confidence.

Problems :

- Only works if the classifiers produce confidence scores on the same scale
- individual binary classifiers see heavily imbalanced data
- one class might not be linearly separable from others.

One-VS-One Approach

Idea: Train n/2 classifiers, one for each pair of classes and apply a voting scheme to predict: the class with the highest number of positive predictions wins.
=> slower but needs no confidence

Multi-Class SVM's

Idea: maintain n weight vectors, one for each class => multi-class hinge loss function.

NEURAL NETWORKS

- Composed of modules called hidden layers
- Able to approximate non-linear functions.

Activation Functions

Idea: used to determine the output of a neural network, mapping values between 0 and 1.

Sigmoid Function

- Exists between 0 and 1.
- it is especially used for models where the expected output is a probability.

Hyperbolic Tangent Function

- like sgmoid but better, because its range is $[-1, 1]$.
- the advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero.

Rectified Linear Unit Function

- most used
- the issue is that all negative values become zero immediately which decreases the ability of the model to fit from the data properly.

Leaky ReLU Function

- under 0 => 0.01...

PART 2 - UNSUPERVISED LEARNING

CLUSTERING

Goal: given a set of data points, group them into clusters such that similar points are in the same cluster and dissimilar points are in different clusters.

Approaches

- **Hierarchical Clustering**: build a tree by starting with each point being in a cluster and then merging the clusters.
- **Partitioning Approach**: optimise cost functions based on partitions.
- **Model-based Approaches**: maintian a model for each cluster and infer membership of new data points from these models.

k-Means Clustering

> Unsupervised analog of the perceptron

Idea: model each cluster by a single point, namely its centre, then assign the point to the closest centre.

Lloyd's Heuristic

Instead of using stochastic gradient descent on this problem, we use another approach:

```
initialize cluster centers
while the algorithm has not converged:
  assign each point xi to the closest center
  update center as mean of the assigned data points
```

Theorem: theLloyd's heuristic algorithm is guaranteeed to monotonically increase the avergae squared distances in each iteration.

Caveats

- does not guarantee to find the global optimum
- it is sensitive to initialisation
- there is no principled way to choose k
- there as no uncertainties about the cluster assignements

Picking k & Model Selection

We cannot use cross-validation since we do not know what to predict in unsupervised learning, hence we do not know if our prediction is good or not.

Heuristic Approach

Often there is parameter k' after which the returns in the loss function are diminishing when increasing f' further => good choice to chose k at this point.

Regularisation

Another approach is to penalise the number of clusters to keep the model simple.

DIMENSIONALITY REDUCTION

Goal: Given a dataset $D = x_1 \dots x_n, D \in \mathbb{R}^d$ obtain a low dimensional representation $\in \mathbb{R}^k, k < d$. This reduction should compress the data but still allow a sufficiently accurate reconstruction.

Linear : Principle Component Analysis

This is a statistical procedure to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables using an orthogonal transformation.

Step 1: Standardisation

Goal: standardise the range of continuous initial variables so that each one of them contributes equally to the analysis, because if there are large differences between the ranges of initial variables , those variablkes with larger ranges will dominate over those with smaller ranges.

How: $z = \frac{value - mean}{standardderivation}$

Step 2 : Covariance Matrix Computation

Goal: understand how the variables of the input data set are varying from the mean with respect. to each other => if there are relationships between them.

How :

- if the covariance is positive then the two variables increase/decrease together
- if the covariance is negative then one decreases when the other increases and vice-versa.

Step 3: Compute the Eigenvectors/Eigenvalues of the Covariance Matrix to identify the Principal Component

Principal Component: new variables constructed as linear combinations of mixtures of the initial variables. These are done in such a way that the new variables are uncorrelated and most of the information of the initial variables is compressed into the fierst components. The compression is done by putting as much information as possible in the 1st component, then as much as possible in the 2nd, until all information is stored.

This enables to reduce dimensionality without losing too much information and discard the components with low information and consider the remaining components as your new variables.

Step 4: Feature Vector

With the remaining components we form a matrix of vectors called Feature Vector

Step 5: Recast the Data along the PC axis

Goal: use the feature vector to reorient the data from the original axes to the ones represented by the principal components .

How: $FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$

Caveats

- PCA is linear , cannot model non-linear structures
- it does not take supervied information into account
- computing via SVD assumes centering of the data

Non-linear: Kernelized PCA

same same but different

???

Non-linear: Auto-Encoders

Goal: learn the identity function f, hence given some data point x we eant to learn f_1, f_2 : $f(x, \theta) = f_2(f_1(x, \theta_1), \theta_2)$.

Use Case: useful for tasks like denoising.

PART 3 - PROBABILISTIC APPROACHES TO ML

PROBABILISTIC MODELLING FOR REGRESSION

Idea: statistically model the data, by quantifying uncertainty and expressing prior knowledge about the data.

Fundamental Assumption: the data set we work on is generated independently and identically distributed

Goal: identify the hypothesis that minimizes the prediction error, which is defined in term of a loss function

Statistical Perspective on Least-Squares Regression

???

Maximum Likelihood Estimation

A method that determines values for the parameters of a model, and they are found such that they maximize the likelihood that the process described by the model that produced the data were actually observed.

How: we want to calculate the total probability of observing all the data.
For a Gaussian distribution : $P(x;\mu,\sigma) = \frac{1}{\mu\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$.
Then perform derivation in order to find the values for μ, σ.

Bias-Variance Tradeoff

Prediction error = $Bias^2$ **+Variance + Noise**.

Let h be the optimal hypothesis when knowing $P(X,Y)$:

- Noise: risk incurred by the optimal model => irreducible : $E_{X,Y}[(Y-h(X)^2)]$
- Variance: risk incurred due to estimating model from limited data : $E_X[E_D[h_D(X) - E_{D'}[h_{D'}(X)]]^2]$
- Bias: excess risk of best model considered compared to minimal achievable risk when knowing $P(X,Y)$: $E_X[E_D[h_D(X)] - h(X)]^2$

Statistical Perspective on Ridge Regression

Idea: Recall that ridge regression was introduced as a way to avoid overfitting when solving linear regression otpimization problems, and now we want. to solve the same problem from a probabilistic perspective.

Use a **Maximum A Posteriori Estimation** : given some prior distribution $P(\Theta)$, we can treat Θ as a random variable and thus we can use Bayes' Theorem to compute the posterior distribution of it.

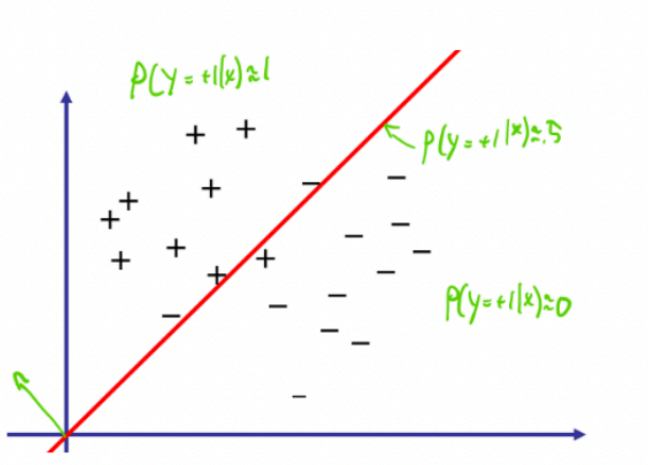
PROBABILISTIC MODELLING FOR CLASSIFICATION: LOGISTIC REGRESSION

For this classification tasks, the risk is different: $R(h) = E_{X,Y}[Y \neq h(X)]$. Hence we want to pick a hypothesis that mminimisesthe risk. First, we want to derive the **Bayes optimal Classifier**, which is given by the most probable class. But because in practice it is unattainable, the natural approach is to estimate $P(Y|X)$.

Logistic Regression

Idea: use a generalised linear model for the class probability.
The goal is that the probability of a point x_i belonging to a class y is somewhat prproportionalo the distance to the decision boundary:

- if it is close to the boundary the probability should be around $1/2$
- it is far it be almost 1 (or 0).



In the case of classification, Y is discrete so we need to uses **Bernoulli noise**. Since the loss function is then convex, we can use Stochastic gradient Descent for the logistic regression.

Regularized logistic Regression

In order to control model complexity we can use a regulariser :

- Gaussian priori is equivalent to regularizing with the L2-penalty: $\lambda||w||_2^2$
- Laplacian priori is equivalent to regularising with the L1-penalty : $\lambda||w||_1$

=> Hence we can use the Maximum A Posteriori Estimation to solve the regularised problem.

Kernel Logistic Regression

Multi-Class Logistic Regression

SVM's VS Logistic Regression

Unlike the other supervised learning approaches that we have looked at , there is no good probabilistic model for SVM's, but they still work often.

- Advantages of SVM's: sometimes higher classifiaction accuracy, solutions are sparse.
- Disadvantages: can't get class probabilities => tis is useful in unclear cases, when SVM simply predicts a point without telling that the prediction is very unclear.

BAYESIAN DECISION THEORY

It is considered to be the ideal pattern classifier and is often used as a benchmark for other algorithms because its decision rule automatically minimizes its loss function.

Derived from the classical probability rule : $P(A|B) = (P(B|A) * P(A)/P(B)$ we obtain for a typical classification task with :

- $A = \omega$: class of an entry
- $B = x$: input feature vector

Hence we get . $P(\omega|x) = (P(x|\omega) * P(\omega)/P(x) \equiv P(\omega|x) = (p(x|\omega * P(\omega))/p(x))$, with :

- $P(\omega|x) =$ the **posterior** which is the probability of the predicted class to be ω for a given entry of feature x.
- $p(x|\omega) =$ class conditional probability density function for the feature. It is called the **likelihood of** ω with respect to x.
- $P(\omega) =$ a **priori probability** of class ω. Depends on external factors and means : how probable the occurence of class ω out of all the classes.
- $p(x) =$ called the **evidence**, a scaling factor that guarantees that the posterior probabilities sum to one.

Decision Rule

For each sample input it calculates its posterior and assigns it to the class corresponding to the maximum value of the posterior probability.

Active Learning

When the data is not given from the beginning but can be extended and requested => active learning.

Uncertainty Sampling: always pick examples that we are mode uncertain about.

1. Compute the probability $p_j = P(y_j = +1|x_j)$ for every unlabelled instance x_j.
2. Introduce an **uncertainty score** u_j and request label ofr instance x_j with $j = argmax_j u_j$

Downside: Active learning violates the fundamental assumption and we can get bad models.

Summary

Learning Through MAP Inference:

1. Place statistical assumptions on data.
2. Choose a likelihood function (=loss function)
3. Choose a priori (= regulariser)
4. Optimize for MAP parameters, i.e. Θ
5. Choose hyper parameters through cross-validation
6. Make predictions via Bayesian Decision Theory

GENERATIVE MODELLING

Logistic regression has a problem : it can be overconfident about labels for outliers. This is because we only estimate the conditional probability but not the probability itself.

Idea: aims to estimate the joint distribution $P(X, Y)$. This is particularly useful since marginal distributions $P(X), P(Y)$ and conditional distributions can be derived from joint distributions but not vice-versa.

Approach: try to infer process according to which examples are generated, i.e. generate labels and then generate features for this class:

1. estimate prior on labels $P(y)$
2. estimate condition distribution for each class y: $P(x|y)$
3. obtain predictive distribution by applying Bayes'rule:
   $P(y|x) = \frac{1}{Z}P(y) * P(x|y) = \frac{1}{Z}P(x, y)$

Naive Bayes Model

Makes the following assumptions:

- labels are generated from categorical varaibles
- features are conditionally independent given Y

Gaussian Naive Bayes Classifier

Further assumption: features are modelled as conditionally independent Gaussians, hence they are only dependent on the class y and the feature i.

Decision Rules for Binary Classification

Discriminant Function: for a binary classifier is defined as the logarithm of the odds : $f(x) = log\frac{P(Y=+1|x)}{P(Y=-1|x)}$ => binary, hence only need to predict the sign of this expression.

Theorem: predicting using the gaussian naive Bayes model is equivalent to logistic regression.

Issues: the conditional independence assumtion is very strong and is violated as soon as there is conditional correlation between classes => can lead to overconfident predictions where the importance of each feature is overestimated. This is only fine if we only care about the most likely class, but if it is used to make predictions, it can be bad.

Gaussian Bayes Model

- Labels are still modelled as generated from categorical variables
- Features are generated by multivariate Gaussians

Idea: we estimate parameters by MLE.

Fisher's Linear Discriminant Analysis

Assume we have binary classification and covariances are equal, this simplifies the discriminant function since the quadratic terms vanish and all is left is a linear classifier. => similar to logistic regression

LDA VS PCA

LDA can be viewed as a projection to a 1-dimensional subspace that maximizes the ratio of between-class and within-class variances. PCA with k=+ in contrast, maximises the variance of the resulting 1-dimensional projection.

- LDA: models joint distribution and can thus be used to detect outliers, but assumes

nprmality of X and is not very good when this assumption is violated
- Logistic Regression: discriminative, cannot detect outliers, but more robust.

Gaussian Naive Bayes VS Gaussian Bayes

- GNB: assuming conditional independence can lead to overconfidence but the number of parameteres is $O(cd)$ and the complexity is linear in d.
- GB: can capture correlation among features and thus avoid overconfidence but it has more parameters and the complexity is quadratic in d.

Categorical Naive Bayes

Difference: Features are modelled as conditionally independent categorical random variables

Generalizing the Bayes Model

Problem: MLE is prone to overfitting => can be combatted by restricting model class which leadds to fewer parameters, or use priors, which leads to more controlled parameters.

Prior over Parameters

Idea to avoid overfitting: place a prior distribution $P(\theta)$ => consider certain classes to be more probable than others and compute posterior distribution $P(\theta|y_1 \dots y_n)$.

Conjugate Distributions: a pair of prior distributions and likelihood functions is called conjugate if the posterior distribution remains in the same family as the prior.

Idea: we can use conjugate priors as regularisers without almost no increase in computational cost, and the hyperparameters can be chosen via cross-validation.

GAUSSIAN MIXTURE MODELS

Goal: Apply generative modelling to unsupervised learning => more specifically when data is missing.

Why generative: try to model how the data was generated => can regenerate the missing features/labels

Gaussian Mixtures

We consider convex combinatins of Gaussian distributions :

$P(x|\Theta) = P(x|\mu, \sum, w) = \sum_{i=1}^{c} w_i N(x; \mu_i \sum_i)$, with $w_i \geq 0, \sum_{i=1}^{c} w_i$

Problem: what we are trying to maximize is non-convex.

Hard Expectation Maximisation

- E-step: predict the most likely class for each data point. After this step we have a complete data set.
- M-step: compute MLE as for the Gaussian Bayes Classifier

Problems with this algorithm :

- points are assigned fixed labels even though the model is uncertain
- if the clusters are overlapping => bad

Soft Expectation Maximisation

If we are given a model $P(z|\Theta) \wedge P(x|z;\Theta)$then we can compute a posterior distribution over cluster membership for each data point => a point can be part of multiple clusters.

Using MLE, one can obtain a system of coupled equations for all parameters we want to optimise over. SInce the system is very hard to solve we use an iterative approach:

- E-Step: calculate cluster membership weights / responsibilities ofr each point x and every cluster j, given the estimates from the previous iteration.
- M-Step: fit clusters to the weighted data points

Special Cases of Gaussian Mixtures

- Spherical: diagonal covariance matrix where all dihgonal entries are the same
- Diagonal: diagonal covariance matrix => conditional independence of features given their respective class => gaussian Naive Bayes
- Tied: all clusters have the same covariance matrix

Paramater Tuning

Initialisation

Initialise weights with uniform distribution, use k-means++ for the means and initialise variances as sperical according to the empirical variance of the data

Selecting k

Cross-validation works well compared to k-means.

Degeneracy

The approach above suffers from: given a single data point x, choosing $\mu = x, \sigma = 0$ the loss converges to $-\infty$. Hence an optimal GMM would chose $k = n$ and put one Gaussian around each data point with vairiance tending to 0, leading to massive overfitting.

=> Solution: add a small term to the diagonal of the MLE to avoid zero variances.

Gaussian-Mixture Bayes Classifiers

We estimate the the class label prior $P(Y)$ and then estimate the conditonal distribution for each class as a gaussian mixture model. Then classify.

Density Estimation & Outlier Detection

Idea: fit gaussian mixture model for density estimation => model $P(x)$ as a gaussian mixture and model the predictive distribution $P(y|x)$ using logistic regression/NN.
=> combines the accurate predictions / robustness of discriminative models with the ability to detect outliers.

How: compare the estimated density of a data point against a threshold, if over the threshhold it is classified as an outlier.

Semi-supervised Learning with GMM's

Obtaining large amounts of unlabelled data is often easy, but getting labelled data is more difficult. With GMM's semi-supervised learning is quite easy.

Theory behind EM-Algorithm

GENERATIVE ADVERSARIAL NETWORKS

Idea: use neural networks in generative modelling to add more flexibility

Implicit Generative Models

Given a sample of unlabelled points, the idea is to have a NN turn a simple distribution into a complicated distribution

Key Issue: a straight-forward approach would be to use MLE on the distribution, but these likelihood functions are very complicated and thus hard to compute and to optimise.

Solution: use surrogates for the likelihood function that are designed for the purpose of density estimation

Generative Adversarial Networks

Idea: simultaneously train two neural networks:

- **Generator G** tries to produce realistc examples
- **Discriminator D** tries to detect fake examples

Trinaing a GAN requires to find a point where the weight of the generator is locally minimal and the weight of the discriminator is locally maximal => saddle point.

=> in reality : need to make sure that the discriminator is not too powerful.

Simultaneous Gradient Descent

Apply SGD to the GAN, the gradients are approximated using samples of minibatches of data points.

Challenges:

- Data Memorisation: may be a degenerate solution
- Mode Collapse: certain modes of the data are very well represented while others are completely ignored.
- Oscillations: no convergence, divergence.

Evaluating GANs: anoither issue that arises is that it is not entirel clear when one has succeeded.