Applied Machine Learning

Saul Pierotti

May 16, 2020

Introduction

- Professor is a physicist in high-energy physics
- We will not go so much into theory
- The exam will be a ML project
- ML is the capacity of a computer to do a task without being explicitly programmed
- AI contains ML, which contains DL (deep learning)
  - ML started in 1980, DL in 2010
- Strong AI is really far
- ML can learn faster and with lower latency than humans
- It is useful for tasks that humans cannot or don't want to do
- Why today? Data avalilable and Cloud computing
- ML can be supervised, unsupervised and reinforcement learning
- Supervised: I know some real solutions
  - It is a regression or classification problem
  - Regression: continuous
  - Classification: discrete
- Unsupervised: no label on the data
  - I use clustering algos
  - I want to find some structure in the data
  - I can get groups, but I don't know the meaning of these groups

Univariate linear regression

- I can define a cost function that measures the average distance of the real outcomes from my regression
- I want to choose the parameters θs that minimize the cost function $J(\theta_1, \theta_2, ..., \theta_n)$
  - In a linear regression the cost function has 2 parameters (!)
    - Intercept and angular coefficient
- To minimise a function I can use a gradient descent algo
  - It is an iterative process
  - For now, only local minima, no global
  - It uses an aggressivness factor α , which is how big every step is
    - If too small it is too slow
    - If it is too large I can miss a minimum
    - α is referred to as an hyperparameter
      - It refers to the learning, not to the problem
  - When updating θs, all of them must be updated simultaneously
- The minimization algo can be analytical or iterative
  - An analytical solution to univariate linear regression exists
  - In ML the analytical version does not scale well
  - GD is the iterative approach

- The iterative update of θ is done by subtracting to its previous value α times the partial derivative of the cost function with respect to θ
    - If the derivative is positive θ decreases, if negative increases, if 0 doesn't change
    - The magnitude of the change is proportional to the derivative at that point (!)
- In a linear regression the cost function is always a convex quadratic: the only minimum is the global minimum (!)
- Batch GD: start from any point and apply GD until I get to a minimum
    - It is batch since at every iteration I evaluate the cost function for the whole batch of datapoints

Multivariate linear regression

- The real world is multivariate (!)
    - Nonetheless, unuvariate is useful for understanding concepts
- I have one θ for each x, plus θ_0
    - θ_0 is a bit unconfortable, since it is different from the others (no x associated!)
    - To make things easier, I introduce $x_0 = 1$ that multiplies θ_0
    - This means that I have n+1 dimentional vectors if n is the number of independent variables
    - In this way, I have a vector of xs and a vector of θs
    - I can represent the whole multivariate function as a product of the x vector with the traspose of the θ vector
    - $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta 2_2 x_2 + ... + \theta_n x_n = \boldsymbol{\theta}^T \boldsymbol{x}$
- The different variables can have different magnitudes, and I want to account for this
    - To correct, I will do feature scaling
    - I divide the data for the highest value for that variable
    - My data becomes all in the range 0-1
    - Outliers can skew my features: I remove them
    - More generally I want to be in the -1/+1 range since x_0 is already 1
    - I need to rescale also features which are really small
- A different way can be to do mean normalisation
    - I subtract the mean and divide for the range (max-min) or stdev

Learning rate

- The selection of α is important for determining if the GD converges, and if it does how much does it take
- How do I determine if the GD has converged?
    - I can decide a threshold decrese, i.e. if J decreses of less than 10^{-3} in one iteration I stop
- If I see a strange behaviour (divergence, bouncing around) the first thing to try is to decrease α
- But what values for α?
    - First try in factor 10 steps: 0.0001, 0.001, 0.01, 1, 10, ...
    - Then go to a factor 3

Polynomial regression

- It is the simplest non-linear model but it can fit really complicated behaviours
- I can create features: instead of using x, why not e^x?
    - I can make linear dependencies which are not linear
- I can reduce any polynomial regression to a linear by adding new features (!)
    - I can use x and x^2 instead of only x

Classification

- Classification problems can be binary or multiclass
- Linear regression is not good for pure classification problems
  - My problem is in nature not linear
  - I want an output in the range 0-1, not a continuous one
- Logistic regression: a classification algorithm
  - It is a sigmoid or logistic function that outputs in the 0-1 range
  - It is a function of the regression function itself $\theta^T x$
    - $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$
  - I can interpret it as a probability of belonging to class y=1, given the measurement x and the parametrization θ
    - $h_\theta(x) = p(y = 1|x, \theta)$
  - In general the logistic function takes any range of values, e.g. outputs of a function, and reports it in the range 0-1
  - The output is the probability of the input belonging to class 1, and the probability of belonging to 0 is its complementary
- I am defining with the logistic a decision bundary that discriminates 1 and 0 outputs
- The decision boundary is not decided by the data, but by our hypothesis
  - It is a product of the model we use
- The decision boundary is not necessarily linear
  - By using higher order polinomials I can have circles and more complex boundaries
- The cost fucntion for the logistic regression cannot be the argument of the logistic
  - If we apply GD on the initial function that is plugged into the logistic, there is no guarantee of convergence
  - This cost function is not convex (!)
- We can define this cost function for a single element y
  - $cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & y = 1 \\ -\log(1 - h_\theta(x)) & y = 0 \end{cases}$
- This can be rewritten as
  - $cost(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$
- The total cost function J is then
  - $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(h_\theta(x_i), y_i)$
  - $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} -y_i \log(h_\theta(x_i)) - (1 - y_i) \log(1 - h_\theta(x_i))$
- The GD algorithm for classification is identical to that for linear regression
  - The only difference is the h itself, so our hypothesis
  - The process for optimizing the descent is the same

Alternatives to GD

- GD is not the only possibility, there is also conjugate gradients and other approaches
- Other approaches are more opaque, there are libraries that provide them but they are difficult to understand

Multiclass classification

- One-vs-all approach: I decompose the problem in several binary classifications
  - I assign a class to 1 and all the other datapoints to 0
  - I determine the decision boundary
  - I repeat with the second class and so on
  - Now we know the probability that a datapoint belongs to each of the classes
  - Our prediction is the class that gives me the highest probability

Overfitting

- I have overfitting when my model does not generalize
- How to reduce overfitting
  - Reduce the number of features
    - This is risky since I can loose useful information
  - Tune down the weight of features
- I don't need to specify how small a feature should be (!)
- I can modify my cost function so that the cost for a feature is really high, and thus gets tuned down by the GD
  - I can add the square of the parameters that I want to tune down to the cost function
  - I add the square because the parameters can be negative (!)
  - In this way I penalise when they get too big
  - This term is added to the sum of squared distances of the previous cost function
- I get a cost function that is a tradeoff between fitting and avoidance of overfitting
- I can do this by introducing the regularization hyperparameter λ
  - It is a multiplier to the sum of squares of all the parameters
    - I am actually excluding θ_0 from this
  - It penalizes the cost function when parameters get too large
- By tuning λ I can modify the behaviour of my model
  - When λ is really large I go towards underfitting
  - When it is too small I have overfitting

Improving performance

- Do not over-optimize the model: if needed try to increase the amount of data
  - Not always easy!
- Another possibility: tune down or remove features!
- Maybe your dataset is not descriptive enough: more features!
- In general, you need experience and gut feeling

Training and testing

- Typically I split in 70/30 or 80/20
- The training set should be larger than the test set
- If there is structure in your data shuffle them!
- When checking performance in the testing set, not always I use the same cost function used in the training
  - For logistic regression I can just count the number of correct predictions
- I can train my model and test it n times, and then chose the one that performs better in the test set
  - In this case I am actually using the test set to choose the best model, so I cannot use it for testing performance!
  - I can 2 a 3-partition: I set aside the test set at the beginning and I do cross validation in the remaining part
    - I choose the best model and then test it on the test set that I set aside
    - A typical split is 60/20/20

Variance and bias

- When my hypothesys is too simple I an doing underfitting, I am in anigh-bias case
- When my hypothests is too complex I an doing overfitting, I am in an high-variance case
- For evaluating bias and variance it is usefull to look at the train error and cross-validation (cv) error

- The train error tends to decrease indefinitely and approach 0 when the degree of the polynomial model increases
- The cv error initially decreases by adding higher polynomial features, but then reaches a minimum and it increases again
- If both train and cv error ar high I am in an high-bias scenario
- If cv error is high and train error low, I am in an high-variance scenario

Automatic regularization

- Choosing a large λ cause high bias, a small λ high variance
- I can automatically select λ by doing model selection
  - I create not 1 but a set of models with a range of λs
  - I test the error on the cv set for each model
  - I choose the λ that minimises J_{cv}

Learning curves

- A learning curve is a plot of J_{train} and J_{cv} as a function of the training set size m
  - I fist train with 1 sample, then with 2, and so on
- J_{train} is small when the set is small and then increases and reaches a plateau when $m \to \infty$
- J_{cv} is high when the set is small and then decreases and reaches a plateau when $m \to \infty$
- The larger m, the closer J_{train} and J_{cv} are
- If J_{train} and J_{cv} are similar I can be in high bias
  - In this case increasing the sample size is not likely to help
- If J_{train} is low but J_{cv} is high (few training samples) I can be in high bias
  - In this case increasing the sample size will help

Classification metrics

- In a classification problem I can define a confusion matrix
  - It has the predicted classes as rows and the actual classes as columns
  - In a 2-class positive/negative scenario $cm = \begin{bmatrix} tp & fp \\ fn & tn \end{bmatrix}$
- Accuracy: general acceptable as a metric but it fails with skewed classes
  - $ACC = \frac{tp+tn}{tp+fp+fn+tn}$
- Precision: fraction of the predicted positives that are actually positives
  - $P = \frac{tp}{tp+fp}$
  - It is the ability of not labeling as negative a positive sample
- Recall: fraction of the true positives that were predicted as positives
  - $R = \frac{tp}{tp+fn}$
  - It is the ability to find all the positve samples
- Precision and recall are resistant to skewed classes: if $tp = 0 \to P = R = 0$
- Usually it is used the convention of assigning the label 1 to the rarest class in a binary classification
  - Precision and recall are evaluated on the rare class (the positives are the rare class)
- In most cases we have a tradeoff between precision and recall
- In some setting I may want to not miss any true positive at the cost of labeling as positive a lot of negatives
  - This can be the case for identifying cancer: I want all the cancer patients to be identified and tested, even if I will mislabel some healty patient that will be tested without need
  - I want high recall at the cost of low precision
- I can plot a PR curve with recall on the x axis and precision on the y axis
  - The curve goes from $x = 0 \to y = 1$ to $x = 1 \to y = 0$

- I can want a single metric that evaluated my algo by combining precision and recall
  - The average of precision and recall is not a good metric
- F1 score: a weighted armonic mean of precision and recall
  - $F1 = 2 * \frac{P*R}{P+R}$
  - It penalizes unbalances between the 2
  - If $P = 0$ or $R = 0 \rightarrow F1 = 0$
  - If $P = R = 1 \rightarrow F1 = 1$
- The F2 score is similar to the F1 score but with different weights
  - $F2 = \frac{5*P*R}{4(P+R)}$
- The ROC curve is a plot of true positive rate (recall) vs false positive rate
  - It is note dependent on a specific threshold
  - $TPR = R = \frac{tp}{tp+fn}$
  - $FPR = R = \frac{fp}{fp+tn}$
  - The ROC curve could be evaluated by testing the model with many different thresholds, but it would be inefficient
  - There is a sorting-based algortihm, AUC that can compute the ROC curve
  - AUC measures the 2d area under the ROC curve
  - A perfect classifier has $AUC = 1$, a random classifier $AUC = 0.5$
  - AUC can be interpreted as the probability that the model will rank a random positive sample higher than a random negative sample
  - AUC is threshold invariant and scale-invariant
  - Scale invariance: it measures how well predictions are ranked, not their absolute values
    - This could be desirable or not depending on context
    - If I want a precise probability output from my model instead of a discrete binary classification, this is not good
  - Threshold invariance: it is a single metric for every threshold
    - This may not be desirable when I want to minimse just 1 kind of error

Neural Networks

- Some times problems are intrinsically non-polynomial and cannot be solved in a traightforward manner with calssic ML algos

Feature crosses

- In many cases NN are useful, but costly in terms of resources and readability
  - When dealing with a lot of data linear learners, if possible, are a better choice
- I can try to attack a complex non-linear problem with feature crosses instead, using standard ML algos
- In general I can take n features $x_1, x_2, ..., x_n$ and combine them in a new feature x_a
  - In this way I can encode non-linearity in the new feature, which will be hopefully linear with the hypothesis!
  - This is transparent for the ML algo, that treats x_a just as another linear feature!
- 1-hot feature vectors: I can have a feature which is actually a vector containing a 1 and all 0s
  - It can encode any kind of information, like class belonging, a binned value, …
  - Binned value: I can subdivide a value range in bins and assign a number in the vecotrs to each of them
    - All the values will be 0 except the one where my sample belongs, which will be 1
- Feature crosses are usually done on 1-hot vectors
  - The result is essentially a logical conjunction
  - If I cross 2 5-class vectors I get a 25-class 1-hot vector encoding the combination of the original vectors
  - There is a possible resulting vector status for each possible combination of the original features

- Take away message: if you don’t have millions of features probably you don’t need a NN
- However, NN are more flexible and can be applied to many cases
- The Google AI playground is a nice visual tool for understanding feature crosses and neural networks

Ensemble learning

- It means combining the predictions of many different algos
- It is one of the most used approach in real ML
- It is probably the approach that will lead to the best performances with classic ML
- There are different ensemble approaches: bagging, boosting, majority voting, stacking
- Bagging: the learners learn independently and the results are then combined in a defined way
  - The classification is done with a simple average of the predictions
- Boosting: I use homogeneous learners that improve the work of each other
  - The output of 1 model is the input of the next one
  - Essentially I scale the features in the way that the model predicts
  - The classification is a weighted average of the predictions
- Majority voting:
- Stacking: an advanced kind of voting that is difficult to implement with sklearn
  - Heterogeneous weak learners are trained independently
  - I create a meta-model that combines the predictions of the various models
- If I have many weak learners with high variance and low bias, bagging is the best choice
- If I have many weak learners with high bias and low variance, boosting is the best choice

Infos for the exam

- For the basic part only scikit learn can be used
- For the advanced part Keras and Tensorflow
- If I want to do a project for both I should use both
- The project is end to end: data and goal
- Option 1: a proposal (1 page or an email)
- Option 2: data and problem given by him
  - There are easy and difficult problems (but they are all quite easy)
- Problems in python with code and documentation, possibly on github
  - It can also be a notebook with both
- It should be reproducible
- You can copy but be clever!