Algoritmeoppgaver som kan komme på eksamen

Pattern matching

Brute Force

Boyer-Moore

Knuth-Morris-Pratt

Tries

Huffman coding

Traveling salesperson

Double Tree Algorithm

Christofides' Algorithm

Minimizing the Sum of Completion Times on a single machine (Chapter 4.1)

MAX SAT

Randomized Algorithm

Derandomization

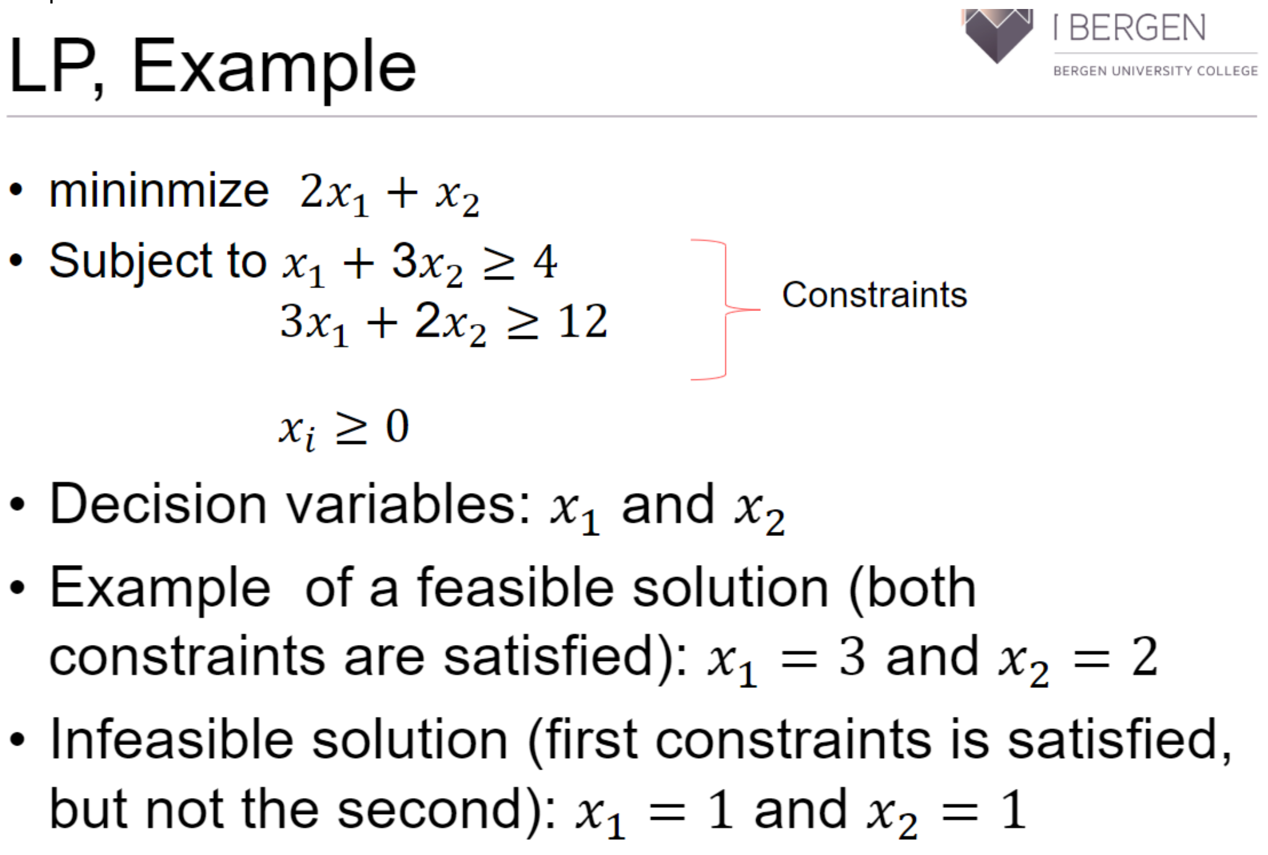
Formulate ILP, LP and Dual given an instance of

Set cover

Linear Programming

- Plays a central role in design and analysis of approximation algorithms
- Important Terms
  - **Decision variables**: represent some sort of decisions that need to be made
  - **Constraints**: Variables are constrained by a number of linear inequalities
  - **Feasible solutions**: Any assignment of real numbers to the variables such that the constraints are satisfied
- Exists efficient algorithms to solve LP

- Example



LP, Example

- mininmize $2x_1 + x_2$
- Subject to $x_1 + 3x_2 \geq 4$
  $3x_1 + 2x_2 \geq 12$ } Constraints
  $x_i \geq 0$
- Decision variables: x_1 and x_2
- Example of a feasible solution (both constraints are satisfied): $x_1 = 3$ and $x_2 = 2$
- Infeasible solution (first constraints is satisfied, but not the second): $x_1 = 1$ and $x_2 = 1$

I BERGEN
BERGEN UNIVERSITY COLLEGE

Integer Programming

- Integer linear programming
  - Exclude fractional solutions
- Zero-One linear programming
  - Allow only 1 and 0 as values for decision variables
- Integer Programming as NP-Hard

Vertex cover

Coloring Graphs(vertices)

2 coloring algorithm

(delta+1) coloring algorithm

the O(sqrt(n)) given Chapter 6.5 (and also in Compulsory 2)

Eksempelspørsmål ML From slide

What is machine learning

Algorithms whose performance improve as they are exposed to more data over time. Performs task without us specifically programming explicit solutions. Machine learning exceeds in difficult to program problems. Where we would struggle to make a well-defined solution. The data might be to complicated or there is simply to much of it. Machine Learning provides a flexible solution. Programs that adapt to their inputs instead of rigid pre-programmed solutions.

Why do a train-val-test split

In ML we're building a model to predict something for new measurements. To simulate having new measurement we need to leave out a set of data when constructing the model. We use this test to test the model and is therefor called a test set.

The rest of the set is train set. This is what we train the model on. The bigger the better for training our model. If it's bigger we get bigger deviation and the model can therefor generalize a bit more. If we use the entire data set the model can typically "remember" the data and cannot predict new data as successfully.

In machine learning we provide Data and answer to the data and it outputs rules for new data in the same category. Det er meir representasjon av features enn data direkte i konvensjonell maskinlæring. I deep learning derimot blir direkte data gitt som input.

Klassisk maskinlæring blir brukt til å finne ting som kreft og lignende.

Three main types of machine learning

1. Supervised learning
    - Learn a model from labeled training data. Used to make predictions on unseen future data.
    - Goal: Learn connections between input and desired output. Function approximation.
    - Example application: will the customer click my ad?
    - 99% of the value generated by ML is based on supervised learning.
2. Unsupervised learning
    - Find meaningful patterns in data, without feedback
    - Clustering, anomaly detection.
    - Example application: Grouping customers based on their purchasing history.
3. Reinforcement learning
    - An agent gathers information from its environment and learns to select actions that maximize a reward.
    - Think: mouse in a maze trying to get the cheese.
    - Example application: AlphaGo

Machine learning slightly with development of important statistical techniques and optimization tools in 1650s. And continued in the 1940s with Cybernetics. Theories of biological learning.

Supervised ML

Some ingredients:

- Data
    - Feature vectors
    - Images
    - Speech
- Labels/annotations
    - {cat, dog, horse}
    - {cancer, !cancer}

    - {coalfish, pollock}
    - {male, female}
- Training data
    - Pairs {(data, label)}
    - Input to ML model
- Trained model
    - A function
        - Sample -> label
- Measure of success
    - Is it doing a good job?
    - Accuracy, loss, ...
    - Used as a feedback signal
- Purpose...
    - What's the model for?
    - Broader context
    - Business impact

Classification

- Cross-Validation
- Confusion matrix
- Precision and recall
- ROC curves
- more stuff

In Classification we want to decide which of the N classes an input belongs to (think of the fish from the slides in the first lecture(racist elmo??????)).

Regression is about finding a function that maps from the inputs to a continous set of numbers:

f: X -> R

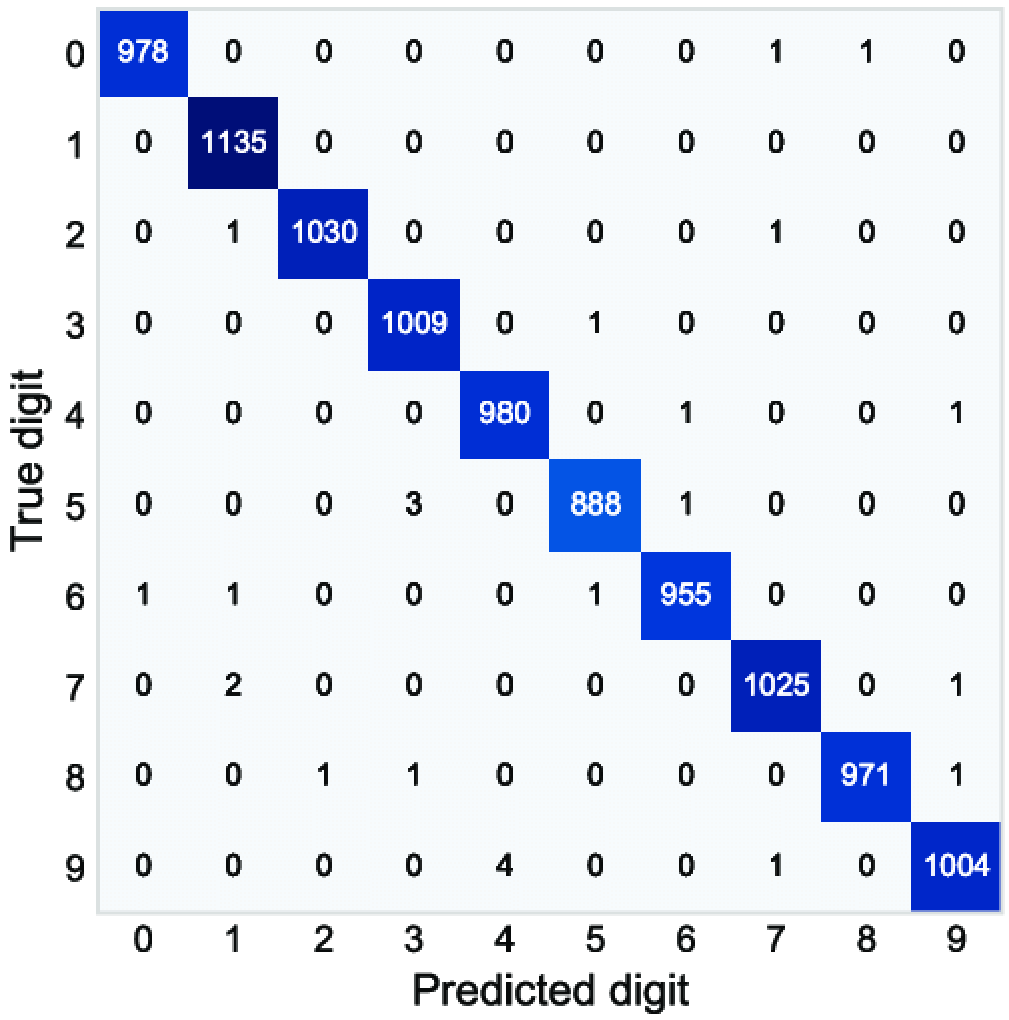
Think of the housing prices predictoin from assignment 1, where you're constructing a function mapping from a set of features of a housing district to the price of a house in that district. Or more generally, say you have set of x-values and corresponding outputs as real numbers. when given a new, previously unsees x-value, what y-value should you predict? What is the best interpolating function

Classification vs Regression

Is there continuity on the possible predictions? Predicting a price of 1 500 000 vs 1 500 001 makes a very small difference

Confusion matrix

Confusion matrix is a table that summarizes the results from a classification model's predictions. The idea is to count the number of times instances of a certain class is classified as the various classes in your problem. So in MNIST case: how often the 4s and non-5s are correctly classified, how often a 5 is misclassified as a non-5 and how often a non-5 as a 5. We need to produce some predictions, this can be done with `cross_val_predict` which returns predictions based on all the K folds.



True digit \ Predicted digit	0	1	2	3	4	5	6	7	8	9
0	978	0	0	0	0	0	0	1	1	0
1	0	1135	0	0	0	0	0	0	0	0
2	0	1	1030	0	0	0	0	1	0	0
3	0	0	0	1009	0	1	0	0	0	0
4	0	0	0	0	980	0	1	0	0	1
5	0	0	0	3	0	888	1	0	0	0
6	1	1	0	0	0	1	955	0	0	0
7	0	2	0	0	0	0	0	1025	0	1
8	0	0	1	1	0	0	0	0	971	1
9	0	0	0	0	4	0	0	1	0	1004

ROC curves

ROC curve(receiver operating characteristic curve) is a graph that's often used to evaluate binary classifiers. It's a plot of the true positive rate vs the false positive rate. TPR = TP/(TP+FN), FNR = FP/(FP+TN). Sklearn has a function for this calles `roc_curve`. A really good classifier will have a ROC curve that's very close to the top left corner. This means it has high TPR and low FPR for any thresholds. We can use ROC curves to compute how good a classifier is. If it's good there's a high area under the curve. If it's bad the area will be close to 0.5. This is called ROCAUC, ROC area under the curve.

What can go wrong if you tune hyperparameters based on test-set performance

The test set is usually a small set and this would cause a lack in generalization. The model would then probably miss a lot on new data.

What is "generalization" in machine learning

Generalization is the ability to predict new data coming in. Data has some general deviation and an overfitted or overtrained model. Or a model trained on a to small set will fail at generalizing new data coming in and miss on it.

What is cross-validation

This is achieved by splitting the training set into several parts called folds. Say K folds. Then we will train the model 3 times. Each time we use a different fold for evaluation and training on the remaining K-1 folds. The average score for the K runs is used to estimate the model's performance. This means that each set is part of the training set K-1 and part of the evaluation set once. An important advantage of this approach over the one above(splitting into two separate sets) is that it doesn't waste as much training data. Unless you have plenty of data, cross validation is the preferred method for estimating model performance.

Cross validatoin also provides a more thorough test than splitting the data into a training set and a test set. The `train_test_split` procedure sets aside a fixed random subset of the data as a test set. If we're unlucky, all the difficult examples end up in the training set, while the test set contains only easy ones.

NOTE: the special case of cross-validation where K is set to the number of data points in the training set is called leave-one-out. Each fold is then a single sample.

The result of this K-fold cross validation procedure is an array of K evaluation scores.

Validatoin set and model selection

It's very important to not base decisions on the parameters in model selection based of test set performance. That is because the model will be biased based on the test set. So to get get an unbiased estimate we split the training set into two sets, a data set used for training and another set for evuualuating performance while trying out various possible models and settings, called a validation set.

Explain concept of "underfitting", "Overfitting" and "generalization" in machine learning

underfitting is when the model is not trained enough and will not hit accurately enough because of purely missing or over generalizing. Overfitting is when the model is trained to much on the selected model and will therefor hit the dataset it's trained on more or less perfectly but will miss on new data. Generalization is mostly what we're after. Generalization in machine learning is essentially how well the model responds to new data coming in. We want a model that generalizes enough to hit new data and not so much that it completely misses.

What is precision? What is recall?

The **recall** of a binary classifier is the proportion of actual positives that were correctly identified. In other words, recall = true positive/all actual positives = TP/P = TP/(TP+FN).

The **precision** of a binary classifier is the proportion of the positive predictions that were actually correct. In other words, precision = true positives/positive prediction = TP/(TP+FP). Sometimes called true positive rate

More precision and recall

Specificity of a binary classifier is the proportion of actual negatives that were correctly identified. Sometimes called true negative rate

Binary classifier

A binary classifier is essentially a system that distinguish one thing from another. Like is this number a five or not a five.

Accuracy and different sorts of errors

If we classify something as belonging to the positive class we can either be correct(**true positive**) or incorrect (**False positive**). If we classify something to be negative we can either be correct (**True negative**) or incorrect (**False negative**). What errors we care the most about depends on the task: if we're for example diagnosing a treatable condition in patients we should do everything we can to reduce the rate of false negatives. While perhaps still keeping an eye on the false positive rate because a positive diagnosis could lead to invasive and extensive further testing of the patients.

In spam filtering we care most about not marking important emails as spam, i.e. we want a low false positive rate(non-spam marked as spam) even if it means a higher false negative rate(some spam emails ending up in our inbox).

What is the so-called precision-recall tradeoff?

It's typically impossible to achieve both high precision and high recall simultaneously. They are typically competing, when one is high the other is low and vice versa.

Are false negatives very bad(like in medical diagnosis)? select a low threshold to get a high recall and OK precision. Are false positives especially costly(like in spam detection)? Go for a threshold that gives you high precision and OK recall.

"If someone says "Let's reach 90% precision", you should ask, "at what recall?""

Training models

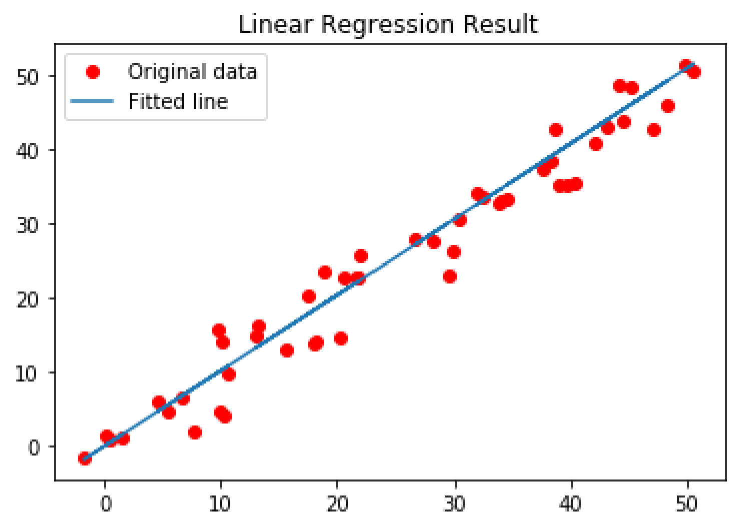
Linear regression

In regression we seek to identify a continuous variable y associated with a given input vector x.

In dumb dumb: We try to find the best straight line through that goes through the points.

y = b + w * x

- y is the dependent variable
- x is the independent variable
- b and w are the constant variables
  - b is called intercept
  - w is called coefficient or slope



Application of linear regression

- Stock market forecast
  - f(past price, size of market, ...etc)= Dow Jones stocks for tomorrow(increase or decrease)
- Self-driving cars
  - f(sensors) = angle of the steering wheel
- Recommendation
  - f(user A, product B) = probability of user A buying product B

Loss-function is sum of square error. We do this to minmize the error to best fit the data points. * How differences between the predicted value and the actual value. * Example: quadratic loss function, absolute loss function, logarithmic loss function, ...etc

Mean square error(MSE): `from sklearn.metrics import mean_squared_error`

Least square method(LSM): `from sklearn.linear_model import LinearRegression`

Root mean square error(RMSE): `import math`

Mean absolute error(MAE): `from sklearn.metrics import mean_absolute_error`

R-squared: `from sklearn.metrics import r2_score`

If we see that the points make a slope or the form of the points is bending a lot we can use polynomial regression.

There's also something called multiple linear regression for when we are comparing more than two variables.

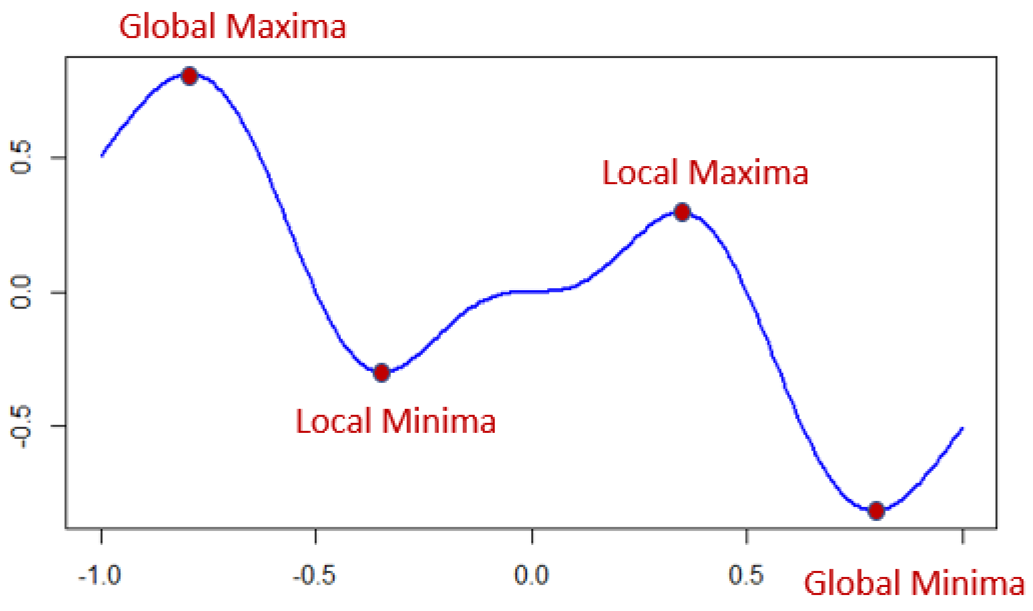
Gradient descent

- An iterative way to find the minimum of the function
- Optimization of function (hyperparameters)
- Optimized the parameters using the gradient descent in the deep learning neural networks
  - Minimum of the function
- Two steps
  - Compute the slope that is the derivative of the cost function at the current point

    - Move in the direction of the slope increase(or decrease) from the current point by the computed amount.

Gradient descent considers a loss function.

- We start by picking an initial random point.
- Compute the differential(slope) of the point to the loss function.
    - If the answer is negative we increase the point
    - If the answer is positive we decrease the point
- We have to consider the learning weight.
    - A too large and constant learning rate will overshoot
    - A small and constant learning rate will take it's goddamn time to reach the minima.
    - A gradient learning rate will hopefully take big steps in the start to get as close as possible and make smaller steps to get as close as possible to the minima without overshooting or undershooting



There can be both local and global minimas.

We terminate the gradient descent either by:

1. Convergence
2. Number of iterations.

Batch gradient descent:

- The parameters are updated once after all the training examples have been passed through the network.
    - 100 training examples then the parameters of the neural network are updated once after all examples are determined.
- **Good at**:
    - It produces a more stable gradient descent convergence and stable error gradient than stochastic gradient descent.
- **Not good at**:
    - Somethimes a stable error gradient can lead to a local minima

- The entire training set can be too large to process in the memory
- It takes too long for processing all the training samples as a batch

Stochastic gradient descent:

- One training sample (example) is passed through the neural network at a time
- Parameters (weights) of each layer are updated with the computed gradient
- We change one-by-one each time

sklearn has an built in function for this: `from sklearn import linear_model`

`clf = linear_model.SGDRegressor(max_iter==10000, eta0=0.0001)`

- **Good at**:
  - It is easier to fit into memory due to a single training sample being processed by the network
    - One sample is processed at each time
  - It can converge faster
    - Updates to the parameters more frequently
  - Help getting out of local minimums of the loss function
- **No good at**:
  - Due to frequent updates the steps taken towards the minima are very noisy
    - This can often lead the gradient descent into other directions
    - Longer achievement

Mini-batch gradient descent:

- This is a mixture of both stochastic and batch gradient descent
  - 100 samples, examines 10, 20, 30..., etc., at each time
- Find the balance between BGD and SGD

A bunch of samples are updated at each iteration

- **Good at**:
  - Easily fits in the memory
  - It is computationally efficient
  - If stuck in local minimums, some noisy steps can lead the way out of them.

Setting learning rate

Learning rate is very important and it is hard to set a good rate. An adaptive rate is good for this. At the beginning, learning rate should be set as a larger rate since we would like to achieve the optimized solution ASAP. After several iterations, we are approaching to the destination, the learning rate should be reduced. Different parameters should have different learning rates.

Adagrad is an adaptive gradient algorithm. Consider to assign the learning rate to each parameter, individually. Learning rate is decreased as the time goes by. As the increase of the steps, we hope learning rate can be converged

Tensorflow is an library full of optimization methods.

Feature scaling is also a thing. Seems to scale down.

Evolution computation computation

- Holland's original GA is now known as the simple genetic algorithm(SGA).
  - Natural inspiration
  - Stochastic search
- Genetic algorithm is used to find the optimized solutions
- Three operations in GA
  - Mutation
  - Crossover
  - Selection

Crossover(same feature with both is the same) rate is much higher than mutation rate(something random)

- Selection
  - Select the best and suitable mate for yourself
    - Fitness function
    - is different in varied domains and applications
- Crossover
  - Marriage for next generation
  - Inherit partial features from parent
    - 30% from dad, 70% from mom
- Mutation
  - Get better solution
    - Become stronger to adapt to the environment

Learning curve

So far:

- Training a model based on the training data
  - Get the parameters
    - LSM
    - Gradient descent
    - ...
  - What is the main purpose
    - Knowing the predicted results of the testing data to evaluate whether this model is good

We need to know the error of the new testing data and find the best function from the function set.

Bias and variance

- **Bias**
  - High bias
    - Pay very little attention to the training data and oversimplifies the model
    - It always leads to high error on training and test set
- **Variance**
  - High variance

        - Pay a lot of attention to the training data and does not generalize on the data which it hasn't seen before
        - Model performs very well on training data but has high error rates on test data

If the model is simple the variance is low, if the model is complex the variance is high. If the model is simple, the result has less influence by the sample data.

- Large bias, underfitting
    - The designed model cannot even fit a training data
- Large variance, overfitting
    - You fit the training data, but has the large error on the testing data
- If the model is underfitting, the model is too simple, the bias is high
    - Re-design the model
        - Add more features into input
            - Weight, height so on
        - A more complex model
            - x squared, x cubed etc.
- If the model is overfitting, the model is already complicated, high variance
    - Add more training data
        - It cannot always be done
    - or regularization

Model selection

There's a trade-off between bias and variance. Select a model to minimize the error in the testing data. How do we select a good model?

Cross validation by splitting into training, validation and testing by using sklearns `train_test_split`. This way we get 60% training, 20% validation and 20% testing.

can also use sklearn `cross_val_score`.

Learning curve

- Learning curves are deemed effective tools for new monitoring the performance of workers exposed to a new task
- Train learning curve
    - Learning curve calculated from the training dataset that gives an idea of how well the model is learned.
- Validation Learning curve
    - Learning curve calculated from a validation dataset that gives an idea of how well the model is generalized.

Problems in machine learning

- Optimization
  - Find the best function to obtain the optimized parameters for the training data.
  - The predicted model cannot be used for all cases
  - Overfitting problem
- Generalization
  - Find the best function to fit all the testing data
  - The parameters cannot be optimized
  - Underfitting problem
- Solution
  1. Start training
  2. High training error?
     - Yes, underfitting high bias
     1. Training longer
     2. Training under more complex model
     3. Obtain more features
     4. decrease regularization
     5. New model
  - High validation error?
    - Yes, overfitting high variance
    1. Obtain more data
    2. Decrease number of features
    3. Increase regularization
    4. New model
  - End

Regularization

- Instead of re-design the model
  - we can do regularization.
- Keep the current features, but reduce the influence of the un-important features
  - Generalization for the coming new data
    - Shrink the importantness of coefficients
  - Good for a model with many features

Ridge regression

- Advantage
  - To solve the problem of overfitting
  - What is overfitting problem?
    - Fit the training data really well but worse in testing
  - Purpose
    - Generalize the model to have better performance on both training and testing
    - Add the penalized term to overcome the overfitting
    - Make the model more generalized, flat, and horizontal
- Loss function
  - LSM (Sum of squared residuals)

Ridge regression line has less slope than the least square line, Less sensitive on X

- Shrinking the coefficients leads to a lower variance and in turn a lower error value
- Decrease the complexity of a model but does not reduce the number of variables, it rather just shrinks their effect
    - It is hard to be 0 for the coefficients, but tends to be a very small number

get a good lambda with trial and error. Try several values for it, use 5-fold or 10-fold cross validation to see which value can make a good results in lower variances

The best situation to use the ridge regression is when the sample data is small, it can make a better prediction to generalize the model for the testing data or prediction is less sensitive to the training data. Ridge regression starts from a small and worse fit to the training data, but has better predictions for the long term

Lasso regression

Absolute values of the slope. less variance than the least square line. less sensitive of X to Y for a small training data. High lambda makes a lower slope.

- **Ridge regression**
    - Shrink the slopes (w_3 and w_4) close to 0
    - Shrink a lot but it is hard to be 0
- **Lasso regression**
    - Shrink the slopes (w_3 and w_4) all the way to 0 if it is not relevant to final results.
    - High possibility shrink to 0

Lasso regression is similar to ridge regression, however given a suitable lambda value lasso regression can drive some coefficients to zero

- Eliminate some features entirely
    - Feature selection

Elastic regression

Lasso can exclude the useless variables, thus reducing the variance in the models if it contains many useless variables.

Ridge is doing better when most variables are useful.

Elastic net is kind of a mix between Lasso and Ridge

Logistic and Softmax regression

Name some features. What are the classes? binary trees

Features in binary trees are usually the difference "categories" one example is level of glucose, BMI and age. In the example from last lecture some features are worst radius, worst concave points, texture error and smoothness error.

The class is somewhat of a result. There are classes in each node, but the class is first assigned in a leaf node.

Say you have an instance with "worst radius" 17, "texture error" 0.5 and "worst concavity" 0.2. What will be the predicted class? What probability will the model assign the instance

False, False, False. This means the predicted class is cancer, but the gini of the class is just 0.48 which is very high.

The probability for not cancer is 2/5 and cancer is 3/5

What is a random forest?

The negative side about binary trees is that it heavily overfits, if not regularly. They are also extremely sensitive to small variations in the data. This is where random trees comes in. We ensemble multiple trees into one model, where one of these is called random forests. Random forest also performs much better. Random forest is related to the concept of wisdom of the crowd where a crowd of non-expert that combine their predictions outperform individual experts.

So to summarize, an ensamble of binary threes which outperforms, is less sensitive and generalizes more than binary threes.

How are predictions from a random forest produced?

This depends on whether it's regression or classification,

- **Regression**: Let each tree make a predicition and then average them.
- **Classification**: Let each tree make a prediction, and use a soft voting strategy to combine them. As we saw in the decision trees notebook, each tree provides a probability for its prediction. Average these probabilities and predict the class that has the highest average probability

What is the idea behind gradient descent

First off gradient descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

Can you name some variants of gradient descent? What are their respective strengths and weaknesses?

- **Batch Gradient Descent**: The parameters are update once after all the training examples have been passed through the network. example: 100 Training examples then the parameters of the neural network are updated once after all examples are determined. **Pros**: Batch is good at producing a more stable gradient descent convergence and stable error gradient than stochastic gradient descent. **Cons**: Sometimes a stable error gradient can lead to a local minima. The entire training set can be too large to process in the memory. It takes too long for processing all the training samples as a batch.
- **Stochastic Gradient Descent**: One training sample is passed through te neural network at a time. parameters(weights) of each layer are updated with the computed gradient. **Pros**: It is easier to fit into memory due to a single training sample being processed by the network. It can converge faster because it updates to the parameters more frequently. Help getting out of local minimums of the loss

function. **Cons**: Due to frequent updates the steps taken towards the minima are very noisy. This can lead the gradient descent into other directions. Longer achievement.

- **Mini-Batch Gradient Descent**: This is a mixture of both stochastic and batch GD. 100 samples, examines 10, 20, 30,... at a time. Find the balance between BDG and SDG. A batch of samples are updated each iteration. **Pros**: Easily fits in the memory. It is computationally efficient. If stuck in local minimums, some noisy steps can lead the way out of them.

What are neural networks? Name some ingredients

Neural networks are set of algorithms that are modeled loosely after the human brain. The algorithms are tasked to recognize patterns. Some ingredients are gradient descent, convolution, pooling, weights, receptive field. The wikipedia explains some categories as building block, these are convolutional layers, pooling layer, ReLu layer, fully connected layer and Loss layer.

Pooling is when we partition the set into partitions into sets of non-overlapping rectangles and for each sub-region outputs the maximum. The idea behind this is that the exact location of a feature is less important than it's rough position. Take the handwritten number as an example, there will be differences for each 0 or 4 and so on. Max pooling is now the most common pooling technique and was until recently the most common one until max pooling recently took over.

Backpropagation is used to calculate how much each weight contributed to the loss by calculating the gradient of the loss with respect to each of them.

Loss layer specifies how training penalizes the deviation between the predicted output and the true labels. There are several types of loss functions for different tasks. Softmax loss is used for predicting a single class of K mutex classes. Sigmoid cross-entropy loss is used for predicting K independent probability values in [0,1]. Euclidean loss is used for regressing to real-valued labels [-infinity, infinity].

Fully Connected Layers are layers where every neuron is connected to every neuron in another layer.

ReLu is the abbreviation on rectified linear unit, which applies the non saturating activation function f(x) = max(0,x). It effectively removes negative values from an activation map by setting them to zero. It increases nonlinear properties of the decision function and of all the overall network without affecting the receptive fields of the convolution layer

Etter svar på melding fra lærer

Neurons

Activation Functions

Organization in layers

Input, output

Can you explain a bit about how neural networks are trained

1. Start with a neural network that has trainable parameters(weights)
2. Collect a batch of training data with corresponding labels, X and y
3. Put a batch X into the network and collect the output predictions y_pred at the output layer.

4. Use the loss function to measure the discrepancy between the known labels y and the predictions y_pred for this batch
5. Update all the weights to reduce the discrepancy for this batch:
    1. Calculate each weight's contribution to the loss, in other words, find the gradients of the weights, by using backpropagation.
    2. Move each weight a little bit in the opposite direction of its gradient(gradient descent). weight = weight - learning_rate * Gradient.
6. Do this over and over for several epochs (one epoch is one pass through all the training data)