Week1 and week2

General

- Generalization ability refers to an algo's ability to give accurate predictions for new, unseen data. Assumptions:
    - Unseen data (like a *test set) should have the same properties as the training set*
    - Models that are accurate on the training set are expected to be accurate on the test set.
- Models too complex with not sufficient training data, are said to **overfit** and are not likely to generalise well to new examples.
- Models that are too simple, can't capture enough nuances of the data and are not likely to generalise well. This is called **under-fitting**.

- The inability for a machine learning method to capture the true relationship is called ***BIAS.***
- The difference in fits between data sets (for example training and test data) is called ***VARIANCE.***
- For example a regression that fits very well training data but very bad testing data, has low bias but high variance.

Scikit-learn flow

1. Cross-validation split.
2. Create model and parameters
3. Fit model to training data
4. Predict using the fitted model on validation or other data.

K-Nearest Neighbours Classification/Regression

- Classification and regression ⟶ Instance based/Memory based supervised learning. It memorises the labels they see in the training. It memorises them to classify later. Makes few assumptions about the structure of the data and gives potentially accurate but sometimes unstable predictions (sensitive to small changes in the training data)

  K=n. Things needed for KNN:

  1. A distance metric (Minkowski distance with power parameter p= 2 = euclidean).
  2. How many nearest neighbours to look at in the feature space.
  3. Optional weighting function on the neighbour points,
  4. How to aggregate the classes of the neighbour points (simple majority votes for example in the case of classification and **mean** in the case of regression).
- Place the new example in the feature space. Calculate the distance (some distance metric) to each of the n nearest neighbour and assign the class of

the majority.

- Low K may result in an overfitted model (and much more sensitive to outliers). So, as we increase K→we (may) decrease the chance of overfitting.
- High K gives a simpler decision boundary = simpler model = decision boundary has much less variance. Single data training points have less influence in the prediction.
- Regression score: [bad;good] → [0;1]. Measures how well a predicion model fits the given data. A value of 0 corresponds to a constant model that predicts the mean value of all training target values. It is also known as *"coefficient of determination"*.

Linear models

- Parameters:
  - w: feature weights/model coefficients
  - b: constant bias/intercept
- Makes strong assumptions about structure of data and gives stable but potentially inaccurate predictions.
- No parameters to control model complexity (always linear).
- Pros:
  - Simple and easy to train.
  - Scales well to large datasets and sparse data.
  - Predictions are easy to interpret.
- Cons:
  - For lower dimensional data, other models may have better generalization.
  - For classification, data may be not linearly separable.

Methods for estimating *w* and *b*

Least-Squares Linear Regression = Ordinary least-squares

- Finds w and b that minimizes the mean square error of the model (sum of squared differences between predicted target and actual target values = RSS).
- The idea is to minimize this cost function:

$$RSS(\mathbf{w}, b) = \sum_{\{i=1\}}^{N} (\mathbf{y_i} - (\boldsymbol{w} \cdot \boldsymbol{x_i} + b))^2$$

Regularizations

https://www.youtube.com/watch?v=9lRv01HDU0s

- The same least-squares criteria, but a penalty for large variations in *w* parameters (=Regularization).
- It penalizes steeper slopes.
- If the fitted line, passes on all training data, the simple cost function is 0. The regulatization adds a value that will make it >0. So now we do need to keep minimizing RSS. The basic idea is to reduce the slope of the fitted line (see video)
- Regularization prevents overfitting by restricting the model, typically reducing complexity.
- Many small/medium sized effects: Use **ridge**.
- If there are many features that may be useless or only a few variables with medium/large effect: Use **lasso**.

Ridge regression

- Ridge regression uses L2 regularization: Minimize sum of squares of w entries.
- The cost function becomes:

$$RSS_{ridge}(\mathbf{w}, b) = \sum_{\{i=1\}}^{N} (\mathbf{y_i} - (\boldsymbol{w} \cdot \boldsymbol{x_i} + b))^2 + \lambda \sum_{\{j=1\}}^{p} w_j^2$$

- The influence of the regularization term is controlled by λ. The higher the more regularization and simpler model.

- The regularization adds bias due to the penalty but decreases variance.
- The ridge regression penalty results in a line with a smaller slope, which means that predictions with ridge regression are less sensitives to *features* that least-squares.
- As alpha (or lambda) increases, the output becomes less sensitive to the *features***.**
- Feature normalization is a must, otherwise some features will contribute more than others.
- Very useful for small sample size compared to the number of features. If there is a lot of training data there may be no need of regularization.

Lazzo regression

- L1 penalty.

$$RSS_{lasso}(\mathbf{w}, b) = \sum_{\{i=1\}}^{N} (\mathbf{y_i} - (\boldsymbol{w} \cdot \boldsymbol{x_i} + b))^2 + \lambda \sum_{\{j=1\}}^{p} |w_j|$$

- Effect of setting parameter weights in w to zero for the least influential variables. This is a sparse solution: a kind of **feature selection**.

Polynomial features with linear regression

- The idea is to transform the data into polynomial and generate new features by combining them all.
- The degree of the polynomial specifies how many variables participate at a time in each new feature.
- This is still a weighted linear combination of features and thus can use least-squares estimation.
- Example for two features and degree 2.
- Adding a high degree feature expansion can lead to complex models that overfit. So often, it is used with regularization.

$$x = (x_0, x_1) \rightarrow x' = (x_0, x_1, x_0^2, x_0 x_1, x_1^2)$$

$$\hat{y} = \hat{w}_0 x_0 + \hat{w}_1 x_1 + \hat{w}_{00} x_0^2 + \hat{w}_{01} x_0 x_1 + \hat{w}_{11} x_1^2 + b$$

Logistic regression

- Used when the target value is categorical (binary or multilabel)
- It is similar to linear regression, with the added logistic function applied to the linear combination.
- It compresses the output between [0;1] which is interpreted as the probability of the input belongs to the positive class.

$$\hat{y} = \frac{1}{1 + e^{-y}}$$

- L2 regularization is 'on' by default on sklearn.

Linear classifier: Support Vector Machine

- A simple model can be a linear one, and then apply sign function to classify a binary label→ f(x,w,b)= sign(w∘x+b)
- The classifier margin is defined as the maximum width the decision boundary area can be increased before hitting a data point.
- The linear classifier with the maximum margin is a linear **support vector machine** (LSVM).
- The parameter C is related to regularization.
  - Larger C values: Less regularization → Fit the training data → few errors as possible → Smaller decision boundary.
  - Smaller C values: More regularization → More tolerant to errors on individual points → Greater decision margin.

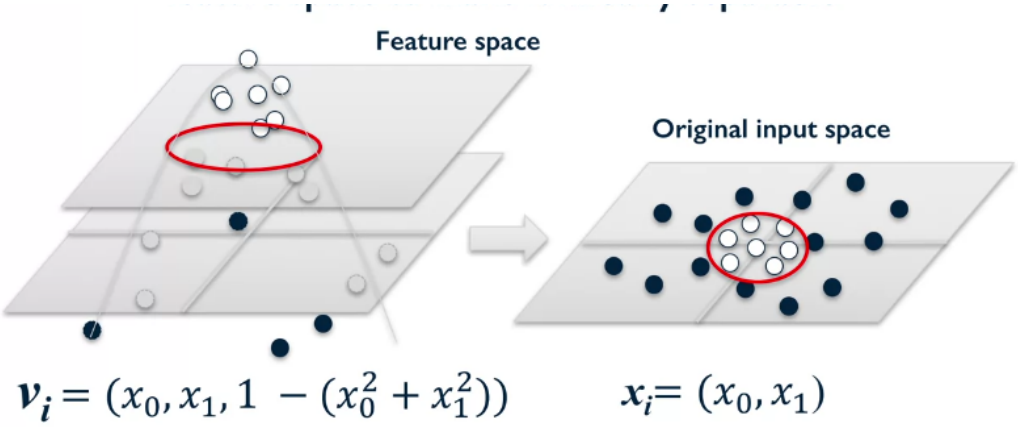
Multi-class classification

- The easy approach is to take the problem and turn it into binary (1 against all other labels).

Kernelized Support Vector Machines

- The Kernelized Support Vector Machines tries to find the decision boundary with maximum margin between classes using a linear classifier in the transformed feature space.
- Useful for classification and regression.
- It can generalize to non-linear data.
- It takes the n features (n dimensions) which can't be separated linearly and turns it into a higher dimensional space, where it can be separated linearly (so it created a non linear in the original space which separated well classes). For example

$$(x_0, x_1) \rightarrow (x_0, x_1, 1 - (x_0^2 + x_1^2))$$



- The kernel is the transformation we apply to the data. It tells us for example, given 2 points in the original input space, what is their similarity in the new feature space.
- In the new feature space, the "N-plane" is set using the maximum margin rule. In the original space, this doesn't hold, and there are various distances obviously.
- **Needs normalization**
- Computation-wise, it doesn't need to transform the dataset, it just calculates the similarity between data pairs.

- For a Radial Basis Function Kernel, the similarity is given by this function:

$$K_{x,x'} = exp[-\gamma \cdot ||x - x'||^2]$$

- Gamma controls how far a single training example reaches. Ends up controlling the width of the decision boundary.
  - Small gamma: Larger similarity radius, so further points are grouped together similar. Smoother decision boundaries.
  - Large gamma: Small similarity radius, so points have to be very close to be considered similar. Complex decision boundaries.

Feature normalization

- Important for some ML methods that all features are on the same scale (faster convergence, more uniform or fair influence for all weights)
- Needed in **regularized regression, K-NN, SVM, NN.**
- The normalization is **calculated** and **applied** for the training set. Then the same normalization is **applied** (only) on the test set. Otherwise there is data ***leakage.***

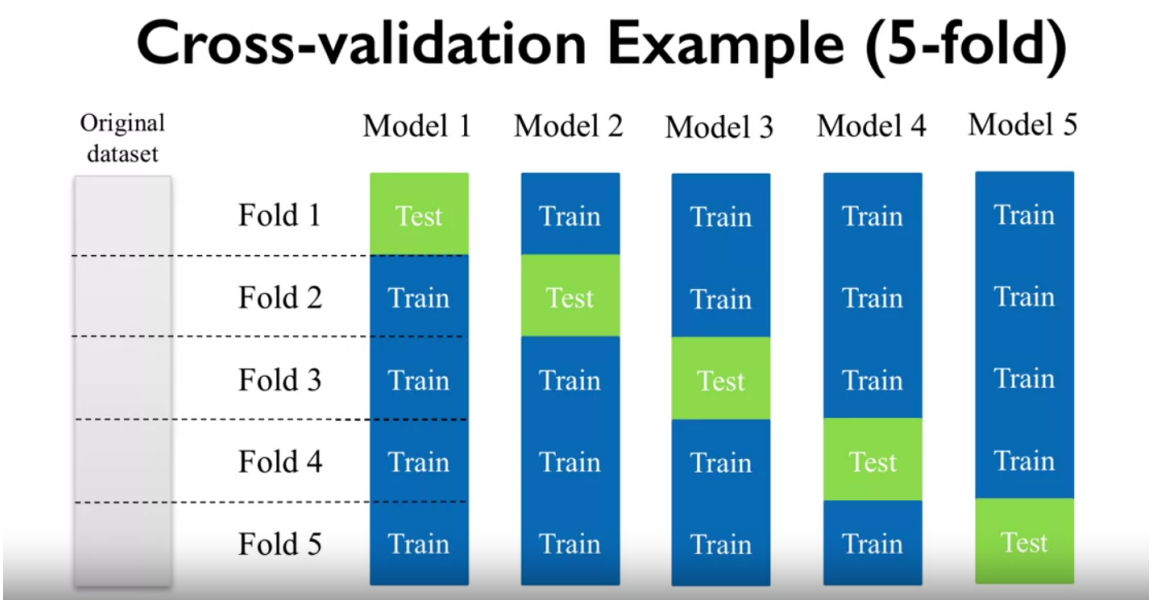
Min-Max scaling → [0;1]

- For each feature, the min and max value is calculated. Then, each feature is transformed using:

$$x_i' = \frac{x_i - x_i^{min}}{x_i^{max} - x_i^{min}}$$

Cross-validation

- The idea is to create multiple train/test splits and fit a new model for each split. Then average the results.
- The proper name is n-fold validation (typically 5 or 10)



- Make sure the dataset is not sorted → **Stratified Cross-Validation**: Makes sure that each split has the same proportion (or tries its best) of classes items.
- Leave-one-out cross validation: Useful for small datasets. Uses n-fold but with all but one element for the training. Then evaluates on a single test sample. This uses more computation because the training set is bigger.
- Validation curve: A sklearn tool that applies n-fold and a range of values to a specific parameter of a model.

Decision trees

- Pros:
  - No need for feature normalization or scaling.
  - Easily visualized and interpreted.
  - it can use different types of features (binary, continuous).
- Cons:
  - Even after tuning, they can overfit.
  - They need an ensemble of trees to generalise better

- For both regression and classification.
- Make binary questions, from the more general to the more granular and providing the best and most informative class split (applying information gain for example, but more on that later)
- Each node is a question, and each answer is a branch
- Each each node gives information like
  - The split point (threshold)
  - Value: Number of samples of each class in that leaf node during training
- When a leaf ends with only one class elements it said to be pure, otherwise it is a mixed node.
- When there are no more splits, the majority of the class in that node, is the resulting class of the sample(s) that ended there. Same goes for inference.
- For regression, the result is the mean of all the training samples that ended in that leaf.
- **It is usual that trees become complex and overfit, so things to overcome this:**
  - sklearn has pre pruning tools for this. In general adjusting only one is enough to reduce overfitting.
    - max_depth: maximum depth (number of split points). Monst common way to reduce complexity.
    - min_samples_leaf: threshold for the minimum # of data instances a leaf can have to avoid firther splitting
    - max_leaf_nodes: limits total number of leaves in a tree
  - Let it grow deep and with pure leafs and then cut some branches (post pruning. not available in sklearn).
- **Feature importance** in a decision tree shows the importance in the early stages of branching for an early separation of classes. A low importance doesn't mean in can be discarded.