**KNN**

**Hyperparameters**:

* number of neighbors (larger k 🡪 smoother surface)
* how the distance between points are computed
* How aggregation over k-neighbors is done
* Majority vote - Weighted vote

Higher k -> lowers the dependence of the model on a particular data point, makes the model more robust and lowers variance

**Decision Tree Algorithms**

* Internal nodes are tests
* A branch in the tree corresponds to test result
* Each leaf node assigns
  + a class : Classification tree
  + Real value : Regression tree

**At each internal test node, how to pick a feature to split on**?

* Split on the feature that reduces **entropy** the most
* Entropy, where is a random variable:
* A chosen feature divides the example set into subsets , according to the distinct values for
* The remaining entropy becomes:
* Information gain

**Pruning** - Discard detailed tests that may use criteria non-essential for classification in test data -> less complex tree, chance of generalizing target function -> less overfitting

**Ensemble Classifiers**

Text, application

Description automatically generated

Text, application

Description automatically generated

**Linear Models**

**Linear Regression**

How well does approximate ?

Squared error:

Training error:

**Normal method for finding regression line:**

* Given a system of linear equations **X**, we want to find the value of weights for the linear regression line such that
* We can find the weights for the linear regression line
* If is not invertible, then we use gradient descent!

1. Construct the matrix **X** and the vector y from the dataset as follows:

,

1. Compute the pseudo inverse
2. Return

**Gradient descent**

**SEE** *Using gradient descent to minimize for logistic regression* for algo

* to minimize loss function (applicable to all loss functions)
* Loss function represents the error between predicted value and actual value
* Move in the direction that decreases loss function
* Magnitude of gradient 🡪 rate of decrease in loss function
* Gradient vector 🡪 direction of descent

|  |  |
| --- | --- |
| **Gradient Descent**  **Many possible termination conditions:**   * Gradient < Threshold * When error change is small * When error is small * When maximum number of iterations is reached | **Stochastic Gradient Descent**  A variation of GD that considers only the error on one data point   * Pick one instance at a time * Apply GD to * **“Average”** direction:     =  =  Benefits of SGD   1. Cheaper computation: Fraction 1/m cheaper per step 2. Stochastic: Helps escape local minima   Cons: More steps, but overall still cheaper computation |

**Logistic Regression**

* Logistic function
* Performs probability prediction on discrete labels
* for logistic regression:

**Using gradient descent to minimize for logistic regression**

1. Initialize the weights at to
2. **Do**
3. Compute the gradient:
4. Update the weights

(moving in the direction of )

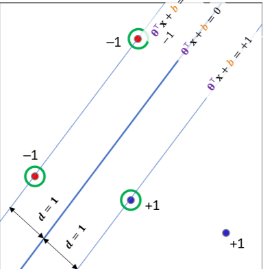
1. Continue to next iteration, until it is time to stop
2. Return the final weights

**Linear regression for classification**

* Linear regression learns a real-valued function
* Binary valued functions are also real-valued!
* Use linear regression to get where
* In this case, is likely to agree with
* Good initial weights for classification

**Noisy Targets – Different outputs despite same inputs**

**Support Vector Machine** – Inherently handles noisy data

Given a hyperplane equation: , +ve points must lie above the hyperplane and negative points below.

* dictates the orientation of the plane,
* dictates the offset from the origin

If we define to be the distance to the hyperplane as ‘1’, this sets up the constrained quadratic optimatization problem that identifies the unique

Note: Only a subset of the dataset determines our unique . These are known as the support vectors

**Linearity -** Linear models work because they are linear in weights 🡪 We can always represent the datapoints in a different vector space, and so long as we can linearly separate the points in some given vector space, we have a linear model.

**Kernels**

A kernel is a function that returns a distance (similarity) measure of two instances:

* K(x,x’) = zTz’ for some Z space.
* Chart, funnel chart

  Description automatically generatedWe can use a suitable kernel to compute similarity for any learning algorithm that measures distance between two points (such as SVM, kNN)

**Soft Margin SVM**

Margin violation if

We can introduce a slack variable :

Total Violation:

**Soft SVM’s loss function:**

Graphical user interface, application

Description automatically generated

* **Scatter chart

  Description automatically generated with medium confidence**Soft margin SVM penalizes misclassifications and correct classifications that fall inside the margin
* In hard margin SVM, by definition, there are no missclassifications
* From the loss function, a higher parameter indicates less tolerance

**Perceptron Learning algorithm**

1. **Diagram, venn diagram

   Description automatically generated**Initialize weights **w**
   * Could be all zero, or random small values
2. For each instance with feature
   * Classify
3. Select one **misclassified** instance
   * Update weights:
4. Iterate steps 2-3 until
   * convergence (classification error < threshold), or
   * **Timeline

     Description automatically generated**max number of iterations

**Gradient Descent weight update (Single neuron)**

**Chain rule for single neuron**

**A picture containing text, clock

Description automatically generated**

**Universal Approximation Theorem**

* Each neuron contributes a piecewise function
* **Text

  Description automatically generated**Many piecewise functions can approximate a curve

**Layer activation**

For a single layer perceptron,

**Text, letter

Description automatically generated**

**Text, letter

Description automatically generated**

**Derivatives for calculating Gradient**

**Graphical user interface, application

Description automatically generated with medium confidence**

**Back propagation**

**Diagram

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

**Text

Description automatically generated with medium confidence 🡨 Recusive**

**Forward Propagation (Reverse Polish Notation)**

**A picture containing diagram

Description automatically generated**

**Lecture 7 – Data preprocessing**

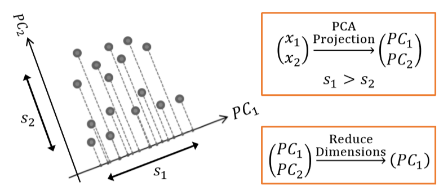
**Dealing with Bad Data**

* **Erroneous Data** aka noise
  + Incorrect feature values or incorrect labelling
  + Some algorithms are robust against noisy data, others more susceptible (e.g. DT w/o pruning is susceptible)
* **Irrelevant Data –** too many irrelevant features impacts performance
  + Apply feature selection
* **Imbalanced Data –** Hard for learning algorithms to cope with imbalanced data
  + Apply oversampling/under sampling
* **Missing Data –** Data collection process was imperfect
  + Apply imputation/interpolation
* **Too many features (high dimensionality)**
  + Data is too sparse in feature space to adequately learn an accurate model
  + Apply feature selection to choose the most relevant features

**Linear Separability**

* If features are not linearly separable, unable to learn a good linear model, need to use more complex models
* Check for linear separability by
  + visualization: 2D scatterplot, or scatterplot matrix for with many features.
  + Computational metrics: Linear SVM – use the sum of all slack variable to quantify the level of violation 🡪 Larger violation = less linearly separable
  + Reduce dimensions with **LDA/PCA**, then check for separability.

**Principal Component Analysis**

* Change the basis of the data set such that data has large variability along the basis. This new basis is known as a principal component.
* We can even eliminate bases by keeping the bases with largest variance.
* Does not use class label info, purely captures the variance in feature values
* **Linear Discriminant Analysis**
* Picks bases such that it maximises distinguishability of classes in the data
* Utilizes class label information.
* Maximizes the mean distance between 2 classes, and minimizes total variance of each group
* Chart

  Description automatically generatedMaximize

**Curse of dimensionality**

* Require exponentially more data instances to maintain density of samples as feature size increases.
* In high dimensional space, most points are nearly the same distance away, causing distance-based learners to break down.
* Check for it by plotting histogram of distances, and check for variance
* Rule of thumb: aim for datapoints > 5x of
* **Mitigation**: Reduce number of features
* **Wrapper methods:** **Recursive feature elimination**
  + Train model **M** with **n** features, eliminate features 1-by-1 and measure decrease in model performance, eliminate feature with least decrease, repeat model **M** with **n-1** features
* **Filter methods:** Information Gain or remove highly correlated features

**Imbalanced Data**

* Note that imbalance here refers to the label. A lopsided distribution of feature values means that the feature is skewed.
* Can be **visualized** by plotting histogram of labels.
* **Mitigated by** Oversampling(SMOTE) Under sampling(Random under sampler)

**Feature Engineering and Extraction**

Process of **transforming raw data** to improve the accuracy of models. Enables you to **capture domain knowledge**, **express non-linear relationships** using linear models and **encode non-numeric features** to be used as inputs to models.

**Tabular Feature Extraction & Engineering**

* Custom non-linear equations based on domain knowledge
* Counting, aggregation, differences, min, max

**Temporal Features**

* Aggregate statistics: Average, variance, range, slope of linear fit
* Sliding time window – Moving average, variance, range, trend

**Image features –** Useconvolutional kernels to extract different features

**Text features**

* Tokenization: Split a sentence into array of substrings (tokens)
* Stemming – Convert words into their base form (word stems)
* Lemmatization – Replace words with their synonyms
* Filtering stop words – Replace common words that have little semantic meaning (e.g. ‘the’, ‘me’, ‘**not’**). But may remove actually informative words.
* **Bag-of-Words** – Preprocess string S to array of keywords, apply one-hot vector for each word in array.
  + **Term frequency – inverse document frequency**  - Identifies words that are unique to the instance. If unique, then word will be more informative to the class label
  + **N-grams** – Instead of BOW, we have bags of phrases
  + **Parts of speech and grammar** - syntax tree to capture structure of language

**Lecture 8 – Bias variance trade-off**

**Bias** – The difference between average prediction and true value.

* How well can approximate overall

**Variance** – The variability of the model prediction, for given data.

* How well can we zoom in on a good

is composed of **Bias** and **Variance**

* Small 🡪 chosen function is good approximation of on unseen data 🡪 hypothesis function fits unseen test data well

**Terminology**

* - -
* – Expected value of , given the distribution of values of
* – Hypothesis of learner when learning from Dataset
* – Expected value of , given distribution of possible Datasets

Focusing on the inner term , define

Therefore,

Application

Description automatically generated with medium confidence

**Approximation-Generalization Tradeoff**

**More complex hypothesis**  🡪better chance of approximating , but runs the risk of approximating the noise (if is low but is high -> overfitting)

**Less complex**  🡪 better chance of generalizing on test data 🡪runs the risk of underfitting, **Ideal = {f}**

**Noise** - Both sources of noise hurt learning:

**Stochastic noise:** Random measurement errors – Changes if we remeasure

**Deterministic noise**: Learner’s canot model – Changes if we change

**Overfitting –** ‘Fitting the data more than is warranted’

* Caused by: stochastic + deterministic noise
* Bias deterministic noise

**Possible solutions for overfitting**:

* **Regularisation**: Restrain the model
* **Validation**: Reality check by peeking at the bottom line

Diagram

Description automatically generated**Validation**

An approximation of

* Split into train, validation and test datasets
* Algorithm:

1. Train on to yield
2. Test on to yield
3. Use cost to estimate
4. Use (not in the end

Rule of thumb – validation set should be 1/5 the size of

Diagram

Description automatically generated**Cross validation**

Given

* : true when validation set is small
* : true when validation set is big

We can estimate many times such that is true

**Evaluation metrics**

* FN – Wrongful positive identification (is it costly to accuse something?)
* FP – Missing out on positive identification (is it costly to miss something? )
* **Accuracy** = (TP+TN)/(TP+TN+FP+FN)
* **Recall**=TP/(TP+FN), **Precision** = TP/(TP + FP), **F1** = 2/(Precision-1 + Recall-1)
* **True Positive Rate** = Recall, **False Positive Rate** = FP/(FP+TN)

|  |  |
| --- | --- |
| **Receiver Operating Characteristic Curve**   * Curve above random line means more accurate than chance | **Area under Curve**   * AUC > 0.5 means better than random chance |
| Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |