Lecture 25: Bagging and Boosting with Decision Trees, Bias-Variance Tradeoff, Feature Selection

Instructor: Prof. Ganesh Ramakrishnan

General Feature Selection based on Gain

- S is a sample of training examples, p_{C_i} is proportion of examples with class C_i in S
- Entropy measures impurity of S: $H(S) \equiv \sum_{i=1}^{K} -p_{C_i} \log_2 p_{C_i}$
- Selecting R best attributes: Let $\mathcal{R} = \emptyset$
- $Gain(S, \phi_i)$ = expected **Gain** due to choice of ϕ_i Eg: Gain based on entropy - $Gain(S, \phi_i) \equiv H(S) \; - \sum_{v \in Values(\phi_i)} \frac{|S_v|}{|S|} H(S_v)$

  Do:
  1. $\phi^* = \underset{\phi_i \notin \mathcal{R}}{\arg\max}\, Gain(S, \phi_i)$
  2. $\mathcal{R} = \mathcal{R} \cup \{\phi^*\}$

  Until $|\mathcal{R}| = R$

General Feature Selection based on Gain

- S is a sample of training examples, p_{C_i} is proportion of examples with class C_i in S
- Entropy measures impurity of S: $H(S) \equiv \sum_{i=1}^{K} -p_{C_i} \log_2 p_{C_i}$
- Selecting R best attributes: Let $\mathcal{R} = \emptyset$
- $Gain(S, \phi_i)$ = expected **Gain** due to choice of ϕ_i Eg: Gain based on entropy - $Gain(S, \phi_i) \equiv H(S) \; - \sum_{v \in Values(\phi_i)} \frac{|S_v|}{|S|} H(S_v)$

```
Do:
    1. $\phi^* = \underset{\phi_i \notin \mathcal{R}}{\arg\max}\; Gain(S, \phi_i)$
    2. $\mathcal{R} = \mathcal{R} \cup \{\phi^*\}$
Until $|\mathcal{R}| = R$
```

Q: What other measures of **Gain** could you think of?

Supervised Feature Subset Selection (Optional)

- One can also Optimally select subset of features using Iterative Hard Thresholding[1] for **Optimal Feature Selection**
- **Input:** Error function $\mathbf{E}(\mathbf{w})$ with gradient oracle to compute $\nabla\mathbf{E}(\mathbf{w})$ sparsity level s, step-size η:
- $\mathbf{w}^{(0)} = 0, \ t = 1$
- while not converged do
  1. $\mathbf{w}^{(t+1)} = P_s\left(\mathbf{w}^{(t)} - \eta\nabla_{\mathbf{w}}\mathbf{E}(\mathbf{w}^{(t)})\right)$ //Projection function $P_s(.)$ picks the highest weighted s features as per the update $\mathbf{w}^{(t)} - \eta\nabla_{\mathbf{w}}\mathbf{E}(\mathbf{w}^{(t)})$ and sets rest to 0
  2. $t = t + 1$
- end while
- Output: $\mathbf{w}^{(t)}$

[1] https://arxiv.org/pdf/1410.5137v2.pdf

Injecting Randomness: Bagging and Ensemble

Main loop:

1. $\phi_i \leftarrow$ **"best" decision attribute for next** *node*
2. Assign ϕ_i as decision attribute for *node*
3. For each value of ϕ_i, create new descendant of *node*
4. **Sort training examples to leaf nodes**..............

Steps (1) and (4) **prohibitive** and **excessively greedy** with large numbers of attributes (1000s) and training examples (100000s). **Alternatives?**

Injecting Randomness: Bagging and Ensemble

Main loop:

1. **$\phi_i \leftarrow$ "best" decision attribute for next *node***
2. Assign ϕ_i as decision attribute for *node*
3. For each value of ϕ_i, create new descendant of *node*
4. **Sort training examples to leaf nodes**..............

Steps (1) and (4) **prohibitive** and **excessively greedy** with large numbers of attributes (1000s) and training examples (100000s). **Alternatives?**

- **Bagging** = **B**oostrap **agg**regat**ing**
- Uniformly at random (with replacements), sample subsets $\mathcal{D}_s \subseteq \mathcal{D}$ of the training data, $\Phi_s \subseteq \Phi$ of the attribute set and construct decision tree T_s for each such random subset.
- Random Forest Algorithm:

Injecting Randomness: Bagging and Ensemble

Main loop:

1. **$\phi_i \leftarrow$ "best" decision attribute for next** *node*
2. Assign ϕ_i as decision attribute for *node*
3. For each value of ϕ_i, create new descendant of *node*
4. **Sort training examples to leaf nodes**..............

Steps (1) and (4) **prohibitive** and **excessively greedy** with large numbers of attributes (1000s) and training examples (100000s). **Alternatives?**

- **Bagging** = **B**oostrap **agg**regat**ing**
- Uniformly at random (with replacements), sample subsets $\mathcal{D}_s \subseteq \mathcal{D}$ of the training data, $\Phi_s \subseteq \Phi$ of the attribute set and construct decision tree T_s for each such random subset.
- Random Forest Algorithm: For $s = 1$ to B repeat:
  1. **Boostrapping:** Draw a random sample $\mathcal{D}_s$ (with replacement) of size m_s from the training data $\mathcal{D}$ of size m
  2. Grow a random decision tree T_s to $\mathcal{D}_s$ by recursively repeating steps (1) - (5) of decision tree construction algorithm,, with following difference to step (1)

Injecting Randomness: Bagging and Ensemble

Main loop:

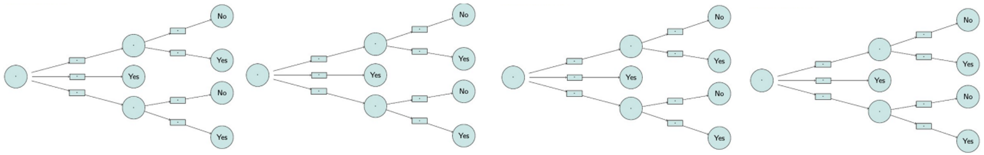
1. $\phi_i \leftarrow$ **"best" decision attribute for next** *node*
2. Assign ϕ_i as decision attribute for *node*
3. For each value of ϕ_i, create new descendant of *node*
4. **Sort training examples to leaf nodes**..............

Steps (1) and (4) **prohibitive** and **excessively greedy** with large numbers of attributes (1000s) and training examples (100000s). **Alternatives?**

- **Bagging** = **B**oostrap **agg**regat**ing**
- Uniformly at random (with replacements), sample subsets $\mathcal{D}_s \subseteq \mathcal{D}$ of the training data, $\Phi_s \subseteq \Phi$ of the attribute set and construct decision tree T_s for each such random subset.
- Random Forest Algorithm: For $s = 1$ to B repeat:
  1. **Boostrapping:** Draw a random sample $\mathcal{D}_s$ (with replacement) of size m_s from the training data $\mathcal{D}$ of size m
  2. Grow a random decision tree T_s to $\mathcal{D}_s$ by recursively repeating steps (1) - (5) of decision tree construction algorithm,, with following difference to step (1)
     1. $\phi_i \leftarrow$ **'best" decision attribute for next** *node* **from** Φ_s **where** $\Phi_s \subseteq \Phi$ **is sample of size** n_s
- **Output:** Ensemble of Trees $\{T_s\}_1^B$

Random Forest applied to Query (Test) data

Output of Random forest Algorithm: Ensemble of **Weakly Learnt** Trees $\{T_s\}_1^B$

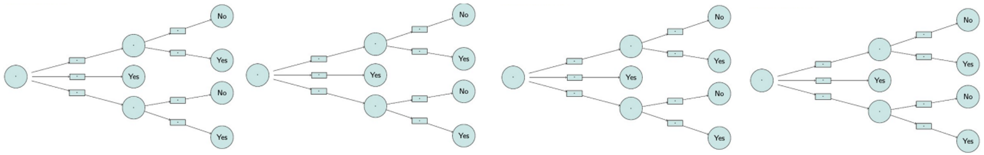


- Consider $\Pr_t(c \mid \mathbf{x})$ for each each **weakly learnt** tree $t \in B$ for each class $c = [1..K]$ based on the proportion of training points in class c of the leaf node determined by the path of query point $\mathbf{x}$ on tree t
- Decision for a new test point $\mathbf{x}$:

[2] Brieman et. al. `http://www.jmlr.org/papers/volume9/biau08a/biau08a.pdf` and `https://www.microsoft.com/en-us/research/publication/decision-forests-a-unified-framework-for-classification-regression-density-estimation-manifol` for several other results on random forests

Random Forest applied to Query (Test) data

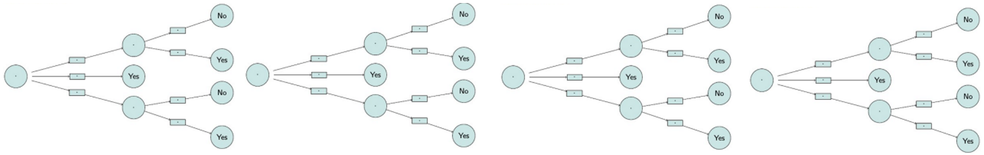
Output of Random forest Algorithm: Ensemble of **Weakly Learnt** Trees $\{T_s\}_1^B$



- Consider $\Pr_t(c \mid \mathbf{x})$ for each each **weakly learnt** tree $t \in B$ for each class $c = [1..K]$ based on the proportion of training points in class c of the leaf node determined by the path of query point $\mathbf{x}$ on tree t
- Decision for a new test point $\mathbf{x}$: $\Pr(c \mid \mathbf{x}) = \frac{1}{|B|}\sum_{t=1}^{B} \Pr_t(c \mid \mathbf{x})$
- For m data points, with $|B| = \sqrt{m}$, consistency results have been proved[2]

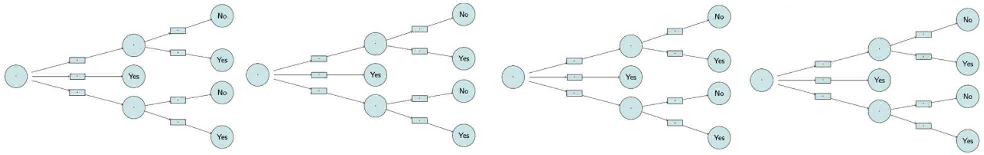
[2] Brieman et. al. http://www.jmlr.org/papers/volume9/biau08a/biau08a.pdf and https://www.microsoft.com/en-us/research/publication/decision-forests-a-unified-framework-for-classification-regression-density-estimation-manifol for several other results on random forests

Random Forest: Balancing Bias and Variance



- Decision for a new test point $\mathbf{x}$: $\Pr(c \mid \mathbf{x}) = \frac{1}{|B|}\sum_{t=1}^{B} \Pr_t(c \mid \mathbf{x})$
- Each single decision tree, viewed as an estimator of the *ideal* tree has high variance, with very less bias (assumptions)
- But since the decision trees T_i and T_j are uncorrelated, when decision is averaged out across them, it tends to

Random Forest: Balancing Bias and Variance



- Decision for a new test point $\mathbf{x}$: $\Pr(c \mid \mathbf{x}) = \frac{1}{|B|} \sum_{t=1}^{B} \Pr_t(c \mid \mathbf{x})$
- Each single decision tree, viewed as an estimator of the *ideal* tree has high variance, with very less bias (assumptions)
- But since the decision trees T_i and T_j are uncorrelated, when decision is averaged out across them, it tends to
  - have low variance
  - be very accurate
  - not overfit

Bias and Variance

- Bias and Variance are two important properties of a machine learning model.
- They help us measure the accuracy of the model and the dependence between the trained model and the training data set. (Q: Is greater dependence good?)
- **Variance** of a model is the variance in its prediction when trained over different training data sets. (Is high variance good?)
- **Bias** of a model is the difference between the expected prediction of the model and the true values which we are trying to predict. (Is low bias good?)
  - Eg: For the Multivariate Gaussian, the Maximum Likelihood estimator of its mean is unbiased, while of its covariance estimator is biased

Bias and Variance

- Bias and Variance are two important properties of a machine learning model.
- They help us measure the accuracy of the model and the dependence between the trained model and the training data set. (Q: Is greater dependence good?)
- **Variance** of a model is the variance in its prediction when trained over different training data sets. (Is high variance good?)
- **Bias** of a model is the difference between the expected prediction of the model and the true values which we are trying to predict. (Is low bias good?)
  - Eg: For the Multivariate Gaussian, the Maximum Likelihood estimator of its mean is unbiased, while of its covariance estimator is biased
  - $\mathbf{E}_{\mathcal{N}(\mu,\Sigma)}\left(\hat{\mu}_{mle}\right) - \mu = 0$ (zero bias)
  - $\mathbf{E}_{\mathcal{N}(\mu,\Sigma)}\left(\hat{\Sigma}_{mle}\right) - \Sigma = \frac{1}{n-1}\Sigma$ (non-zero bias)
- One can quantify the trade-off between bias and variance. Eg:
  - Expected squared loss error $=$ *variance* $+$ *bias*^{2} $+$ *noise* (see optional slides for details)

Bias and Variance

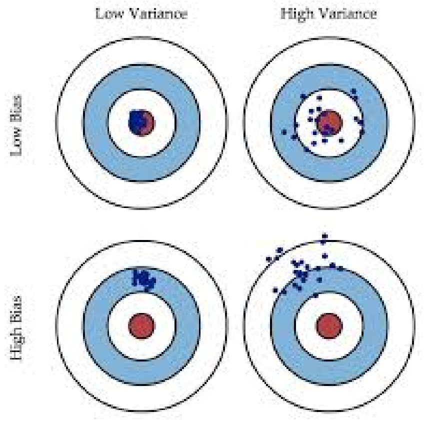
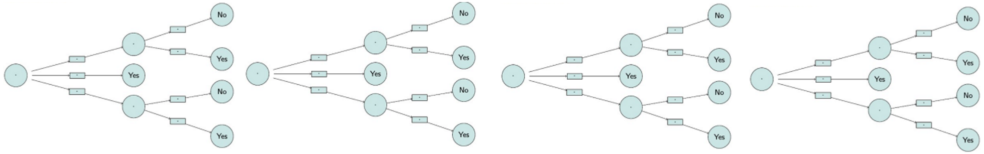


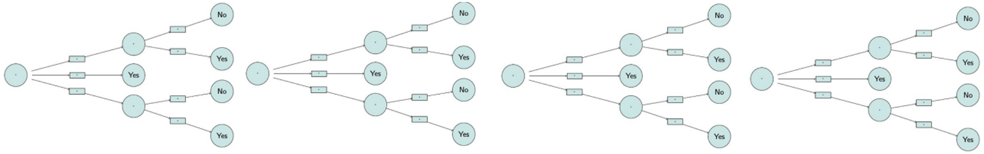
Figure: The distance of the cluster from the eye represents bias and the spread of the cluster represents variance.
(*src: zhangjunhd.github.io/2014/10/01/bias-variance-tradeoff.html*)

Weak Models: From **Bagging** to **Boosting**

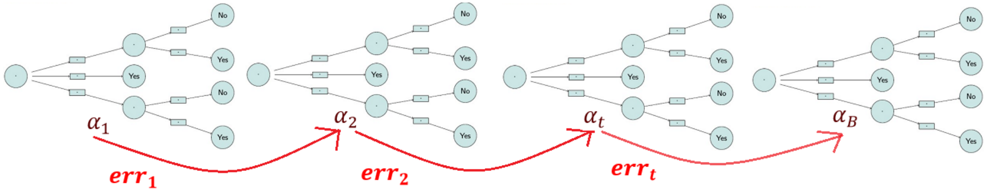


Bagging: Ensemble of **Independently Weakly Learnt** Models (Eg: Trees $\{T_s\}_1^B$):
$\Pr(c \mid \mathbf{x}) = \frac{1}{|B|}\sum_{t=1}^{B} \Pr_t(c \mid \mathbf{x})$

Weak Models: From **Bagging** to **Boosting**

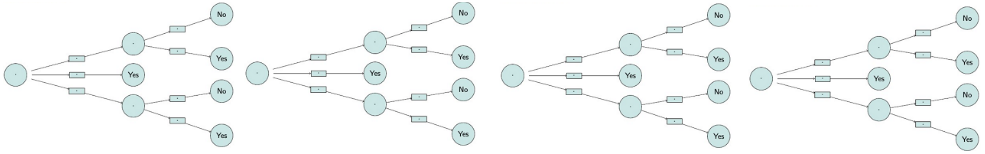


Bagging: Ensemble of **Independently Weakly Learnt** Models (Eg: Trees $\{T_s\}_1^B$):
$\Pr(c \mid \mathbf{x}) = \frac{1}{|B|}\sum_{t=1}^{B} \Pr_t(c \mid \mathbf{x})$

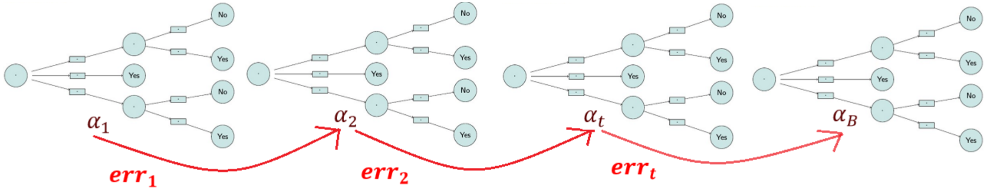


Boosting: Wtd combinations of **Iteratively Weakly Learnt** Models (Eg: Trees $\{\alpha_t, T_t\}_1^B$):

Weak Models: From Bagging to Boosting

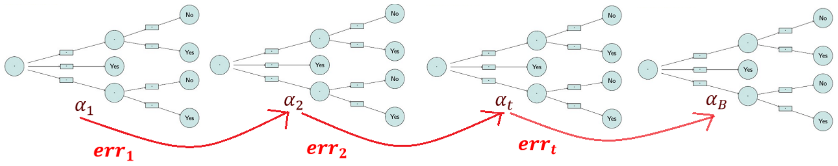


Bagging: Ensemble of **Independently Weakly Learnt** Models (Eg: Trees $\{T_s\}_1^B$):
$\Pr(c \mid \mathbf{x}) = \frac{1}{|B|}\sum_{t=1}^{B} \Pr_t(c \mid \mathbf{x})$



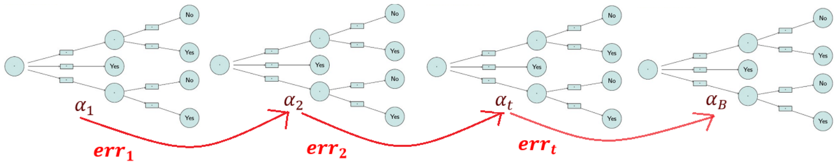
Boosting: Wtd combinations of **Iteratively Weakly Learnt** Models (Eg: Trees $\{\alpha_t, T_t\}_1^B$):
$\Pr(c \mid \mathbf{x}) = \frac{1}{|B|}\sum_{t=1}^{B} \alpha_t \Pr_t(c \mid \mathbf{x})$ where $\alpha_t = (1/2 \ln\left((1-err_t)/err_t\right)$

Adaptive Boosting of Iteratively Learnt Weak Models

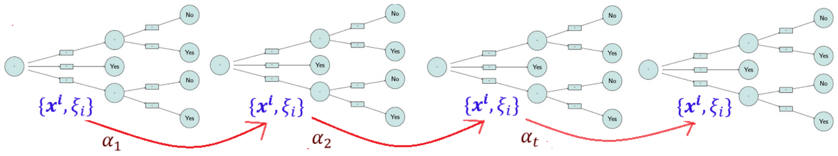


Error driven weighted linear combinations of models: $\alpha_t = (1/2)\ln\left((1 - err_t)/err_t\right)$

Adaptive Boosting of Iteratively Learnt Weak Models



Error driven weighted linear combinations of models: $\alpha_t = (1/2)\ln\left((1 - err_t)/err_t\right)$



Reweighting of each data instance $\mathbf{x}^{(i)}$ before learning the next model T_t:

$$\xi_i = \xi_i \exp\left(\alpha_t \delta\left(y^{(i)} \neq T_t\left(\mathbf{x}^{(i)}\right)\right)\right).$$ Note that $err_t = \frac{\sum_{i=1}^{m} \xi_i \delta\left(y^{(i)} \neq T_t\left(\mathbf{x}^{(i)}\right)\right)}{\sum_{i=1}^{m} \xi_i}$

Adaboost Algorithm

Initialize each instance weight $\xi_i = \frac{1}{m}$. For $t = 1$ to B do:

1. Learn the t^{th} model T_t by weighing example $\mathbf{x}^{(i)}$ by ξ_i
2. Compute the corresponding error on the training set $err_t = \frac{\sum_{i=1}^{m} \xi_i \delta\left(y^{(i)} \neq T_t\left(\mathbf{x}^{(i)}\right)\right)}{\sum_{i=1}^{m} \xi_i}$
3. Compute the error driven weighted linear factor for T_t: $\alpha_{\mathbf{t}} = (1/2) \ln\left((1 - err_t)/err_t\right)$
4. Reweigh each data instance $\mathbf{x}^{(i)}$ before learning the next model:

$$\xi_i = \xi_i \exp\left(\alpha_{\mathbf{t}} \delta\left(y^{(i)} \neq T_t\left(\mathbf{x}^{(i)}\right)\right)\right).$$

Adaboost Algorithm: Motivation (Tutorial 9)

- Freund & Schapire, 1995: Converting a “weak” PAC[3] learning algorithm that performs just slightly better than random guessing into one with arbitrarily high accuracy.
- Let $C_t(\mathbf{x}) = \sum_{j=1}^{t} \alpha_j T_j(\mathbf{x})$ be the boosted linear combination of classifiers until t^{th} iteration.
- Let the error to be minimized over α_t be the sum of its exponential loss on each data point,

$$\mathbf{E}_t = \sum_{i=1}^{m} \delta\left(y^{(i)} \neq sign\left(C_t\left(\mathbf{x}^{(i)}\right)\right)\right) \leq \sum_{i=1}^{m} \exp\left(-y^{(i)} C_t\left(\mathbf{x}^{(i)}\right)\right)$$

- Claim1: The error that is the sum of exponential loss on each data point is an upper bound on the simple sum of training errors on each data point
- Claim2: $\alpha_t = (1/2)\ln\left((1 - err_t)/err_t\right)$ actually minimizes this upper bound.
- Claim3: If each classifier is slightly better than random, that is if $err_t < 1/K$, Adaboost achieves zero training error exponentially fast

[3] `http://web.cs.iastate.edu/~honavar/pac.pdf`

Extra Reading: Bias Variance Trade-off

Instructor: Prof. Ganesh Ramakrishnan

Expected loss of a model

- Say, we are given the training data T_D containing values for x and the target variable is y. $P(x,y)$ is the joint distribution over x and y. $f(x)$ is our target function (as this function will be dependent on T_D as well it is more appropriate to call it $f(x, T_D)$).
- To find the expected loss of the model over the distribution of the training data, we first simplify the expected loss expression. For square loss we get,

$$E_{P(x,y)}[(f(x) - y)^2] = \int_x \int_y (f(x) - y)^2 P(x,y)\,dxdy$$

$$
\begin{aligned}
&E_{P(x,y)}[(f(x)-y)^2] \\
&= \int_x \int_y (f(x)-y)^2 P(x,y)\,dxdy \\
&= \int_x \int_y (f(x)-E(y/x)+E(y/x)-y)^2 P(x,y)\,dxdy \\
&= \int_x \int_y (f(x)-E(y/x))^2 P(x,y)\,dxdy + \int_x \int_y (E(y/x)-y)^2 P(x,y)\,dxdy \\
&\quad + 2\int_x \int_y (f(x)-E(y/x))(E(y/x)-y) P(x,y)\,dxdy
\end{aligned}
$$

We will rewrite the 3rd term in the final equation as:

$$
\begin{aligned}
&2\int_x \int_y (f(x)-E(y/x))(E(y/x)-y) P(x,y)\,dxdy \\
&= 2\int_x (f(x)-E(y/x))\left(\int_y (E(y/x)-y) P(y|x)\,dy\right) P(x)\,dx
\end{aligned}
$$

By definition $\int_y yP(y|x)dy = E(y/x)$. Therefore the inner integral is 0.

Finally we get,

$E_{P(x,y)}[(f(x) - y)^2] = \int_x \int_y (f(x) - E(y/x))^2 P(x,y) dxdy + \int_x \int_y (E(y/x) - y)^2 P(x,y) dxdy$

The 2nd term is independent of f. Can you think of a situation when the 2nd term will be 0?
Q: For what value of f will this loss be minimized?

The minimum loss will be achieved when $f(x) = E(y/x)$
Now let us find the expected loss over the training data. Using our previous analysis we see that only the $(f(x) - E(y/x))^2$ component can be minimized. (Remember f is dependent on T_D)
(Simple Q: Why is integrating over T_D and (x, y) the same)

$$\int_{T_D} (f(x, T_D) - E(y/x))^2 P(T_D) dT_D$$
$$= E_{T_D}[(f(x, T_D) - E_{T_D}[f(x, TD)] + E_{T_D}[f(x, TD)] - E(y/x))^2]$$
$$= E_{T_D}[(f(x, T_D) - E_{T_D}[f(x, TD)])^2 + (E_{T_D}[f(x, TD)] - E(y/x))^2$$
$$-2(E_{T_D}[f(x, TD)] - E(y/x))(f(x, T_D) - E_{T_D}[f(x, TD)])]$$

The last term vanishes (WHY?) and we get:
$$E_{T_D}[(f(x, T_D) - E_{T_D}[f(x, TD)])^2] + (E_{T_D}[f(x, TD)] - E(y/x))^2$$

Bias and Variance

$E_{T_D}[(f(x, T_D) - E_{T_D}[f(x, TD)])^2] + (E_{T_D}[f(x, TD)] - E(y/x))^2$
$= Variance + Bias^2$
Finally we say the expected loss of the model is:
$Variance + Bias^2 + Noise$

The noise in the measurement can cause errors in prediction. That is depicted by the third term.

Interpret with example - Linear Regression

If we were to take the linear regression with a low degree polynomial, we are introducing a bias that the dependency of the predicted variable is simple.
Similarly when we add a regularizer term, we are implicitly biased towards weights that are not big.
By being biased towards a smaller class of models the predicted values will have smaller variation when trained over different samples (Low Variance) and may fit poorly as compared to a complex model (High Bias).
The low variance makes model generalizable over the samples.

Interpret with example - Linear Regression

Suppose we complicate our regression model by increasing degree of the polynomial used.
As we have seen before this will lead to complex curves and will tend to pass through all points.
Here we have put fewer restrictions on our model and hence have less bias.
For a given training data our prediction could be very good (Low Bias).
Although if we consider different Training Sets are models could vary wildly (High Variance).
This reduces the generalizability of the model.

Conclusion

This is the Bias-Variance Tradeoff in action. Simple models usually have low variance but high bias and complex models usually have high variance and low bias.

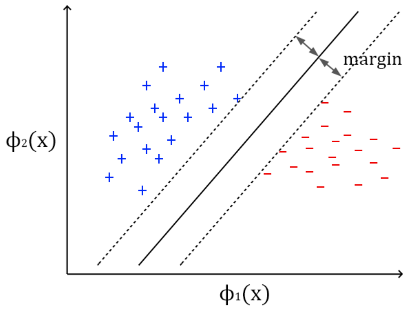
Food for Thought: So how should we choose our model?

Also whenever you learn about a new algorithm it would be a good exercise to see how the tradeoff works there.

For example, think how the tradeoff manifests itself in the K Nearest Neighbor algorithm.

Support Vector Machines

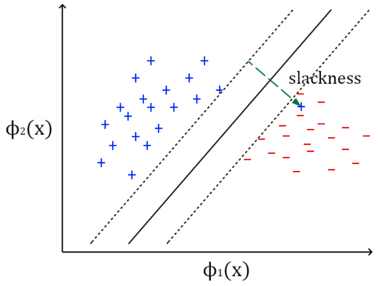
- Perceptron does not find the *best* seperating hyperplane, it finds *any* seperating hyperplane.
- In case the initial w does not classify all the examples, the seperating hyperplane corresponding to the final w^* will often pass through an example.
- The seperating hyperplane does not provide enough breathing space – this is what SVMs address!



$$w^\top \phi(x) + b \geq +1 \ \textit{for}\ y = +1$$
$$w^\top \phi(x) + b \leq -1 \ \textit{for}\ y = -1$$
$$w, \phi \in \mathbb{R}^m$$

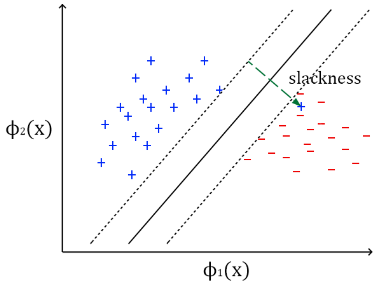
There is large margin to seperate the $+$ve and -ve examples

Overlapping examples



When the examples are not linearly seperable, we need to consider the slackness ξ_i of the examples x_i (how far a misclassified point is from the seperating hyperplane, always $+$ve):

Overlapping examples



When the examples are not linearly seperable, we need to consider the slackness ξ_i of the examples x_i (how far a misclassified point is from the seperating hyperplane, always +ve):
$w^\top \phi(x_i) + b \geq +1 - \xi_i$ (*for* $y_i = +1$)
$w^\top \phi(x_i) + b \leq -1 + \xi_i$ (*for* $y_i = -1$)

Multiplying y_i on both sides, we get:
$y_i(w^\top \phi(x_i) + b) \geq 1 - \xi_i, \forall i = 1, \ldots, n$

Maximize the margin

- We maximize the margin given by $(\phi(x^{+}) - \phi(x^{-}))^{\top}[\frac{w}{\|w\|}]$
- Here, x^{+} and x^{-} lie on boundaries of the margin.

Maximize the margin

- We maximize the margin given by $(\phi(x^+) - \phi(x^-))^\top[\frac{w}{\|w\|}]$
- Here, x^+ and x^- lie on boundaries of the margin.
- Verify that w is perpendicular to the seperating surface:
  at the seperating surface, the dot product of w and $\phi(x)$ is 0 (with b captured), which is only possible if w and $\phi(x)$ are perpendicular.
- We project the vectors $\phi(x^+)$ and $\phi(x^-)$ on w, and normalize by w as we are only concerned with the direction of w and not its magnitude.

Simplifying the margin expression

- Maximize the margin $(\phi(x^+) - \phi(x^-))^\top [\frac{w}{\|w\|}]$
- At x^+: $y^+ = 1$, $\xi^+ = 0$ hence, $(w^\top \phi(x^+) + b) = 1$ — $\textcircled{1}$
  At x^-: $y^- = 1$, $\xi^- = 0$ hence, $-(w^\top \phi(x^-) + b) = 1$ — $\textcircled{2}$
- Adding $\textcircled{2}$ to $\textcircled{1}$,
  $w^\top(\phi(x^+) - \phi(x^-)) = 2$
- *Thus, the margin expression to maximize is:* $\frac{2}{\|w\|}$