CS 6350 Final Exam Review (Notes)

PAC Learning

- Instance Space: $\{X\}$, the set of examples
- Concept Space: $\{C\}$, the set of possible target functions: $f \in C$ is the hidden target function.
  - *Example:* All n-conjunctions; all n-dimensional linear functions,...
- Hypothesis Space: $\{H\}$, the set of all possible hypotheses that the learning algorithm explores
- Training Instances: $Sx\{-1, 1\}$, Positive and negative examples of the target concept (S is a finite subset of x)
- Want Hypothesis $h \in H$ such that $h(x) = f(x)$

Error of hypothesis: Given a distribution $\mathcal{D}$ over examples, the *error* of a hypothesis h with respect to a target concept f is $err_{\mathcal{D}}(h) = Pr_{x\sim\mathcal{D}}\left[h(x) \neq f(x)\right]$.
Empirical error: Contrast true error against emperical error. For a target concept f, the emperical error of a hypothesis h is defined for a training set S as the fraction of examples $x \in S$ for which functions $f \neq h$, denoted by $err_s(h)$.
Overfitting: When the emperical error on the training set $err_s(h)$ is substancially lower than $err_{\mathcal{D}}(h)$.

Batch Learning

Goal: To devise good learning algorithms that avoid overfitting

- Not fooled by functions that only appear to be good because they explain the training set very well

Online Learning
- No assumptions about the distribution of examples
- Learning is a sequence of trials
  - Learner sees a single example, makes prediction. If there is a mistake, then update the hypothesis
- *Goal:* To bound the total number of mistakes

Batch Learning
- Examples are drawn from a fixed (perhaps unknown) probability distribution $\mathcal{D}$ over the instance space
- Learning uses a training set S, drawn from the distribution $\mathcal{D}$
- *Goal:* Find hypothesis that has a low chance of making a mistake on a new example in $\mathcal{D}$

Suppose we are learning a conjunctive concept with a dimensional boolean feature using m training examples. If

$$m > \frac{n}{\epsilon}\left(\log(n) - \log(\delta)\right) \tag{1}$$

then with probability $> 1 - \delta$, the error of the learned hypothesis $err_{\mathcal{D}}(h)$ will be less than ϵ
Proof
We can say that $p(z)$ is the probability that after training z causes a mistake. The *Union Bound* states: For a set of events, the probability that at least one happens is less than the sum of the probabilities of random events. Using this, we can get a bound such as

$$err_{\mathcal{D}}(h) \leq \sum_{z\in h} p(z) \tag{2}$$

Where we call the literal z bad if $p(z) > \frac{\epsilon}{n}$ where n is the dimensionality. Without loss of generality, we can let z be a bad literal. The probability that it will not be eliminated by one training example is defined as

$$Pr(z \text{ survives one example}) < 1 - Pr(z \text{ eliminated by one example}) \tag{3}$$
$$1 - \frac{\epsilon}{n} \tag{4}$$

which is equivalent to the probability that z isn't eliminated for that one example. This can be generalized for m examples via multiplication of probabilities, which results in

$$Pr(z \text{ not elminated for } m \text{ examples}) < \left(1 - \frac{\epsilon}{n}\right)^m \tag{5}$$

where we can have *at most* n bad literals, which results in

$$Pr(\underline{\text{Any}} \text{ bad literals survives an example}) < n\left(1 - \frac{\epsilon}{n}\right)^m \tag{6}$$

We want to minimize the probability in Equation (6), as we want the probability that any z survives all of them is less than δ

$$n\left(1 - \frac{\epsilon}{n}\right)^m < \delta \tag{7}$$

where we know from Taylor Series Expansion $e^{-x} = 1 - x + \frac{x^2}{2!} - \cdots$. We can use a first order approximation to approximate our quantity as being exponential, resulting in

$$ne^{-\frac{m\epsilon}{n}} < \delta \tag{8}$$

$$\text{solve for } m \to m > \frac{n}{\epsilon}\left(\log\left(\frac{n}{\delta}\right)\right) \quad \square \tag{9}$$

If m has this property, then we can say

- Probability of no bad literals: $1 - \delta$
- *i.e.* with probability $1 - \delta$, we will have $err_{\mathcal{D}}(h) < \epsilon$

Realistic Expectation of a Good Learner: With a high probability, the learner will learn a close approximation to the target concept
PAC Learning:

- Given small parameters δ, ϵ, there is a probability of $1 - \delta$ a learner produces $h(x)$ with an error of at most ϵ

Consistent Distribution Assumption: Only way we can do this. We assume that the data in $\mathcal{D}$ is of a consistent distribution throughout the data and it isn't built by multiple distributions
We also need to ipose two limitations:

- *Polynomial* <u>*sample complexity*</u>:
  - Is there enough info in the sampel to distinguish the hypothesis h that approximates f?
- *Polynomial* <u>*time complexity*</u>:
  - Is there an efficient algorithm to sample the data and produce h?

Occam's Razor

Claim: The probability $h \in H$ that:

- Is consistent (yet bad) with m examples
- Has error $err_{\mathcal{D}}(h) > \epsilon$

is less than $|H|(1 - \epsilon)^m$
Proof: Let h be such a bad hypothesis with error$> \epsilon$. The probability that h is consistant with one exapmle is $Pr(f(x) = h(x)) < 1 - \epsilon$.

The given training set consists of m examples drawn independently so, the probability that h is consistent with m examples$< (1-\epsilon)^m$. From this, the probability that a bad hypothesis$\in H$ is consistent with no examples is $|H|(1-\epsilon)^m$. We want to bound this and make it small, so we say

$$|H|(1-\epsilon)^m < \delta \tag{10}$$
$$\ln(|H|) + m\ln(1-\epsilon) < \ln(\delta) \tag{11}$$

where we can again use a first order Taylor Series Approximation of e^{-x} on $(1-\epsilon)$, and then solve for m which gives

$$m > \frac{1}{\epsilon}\left(\ln(|H|) - \ln(\delta)\right) \tag{12}$$

where the probability of a hypothesis $h \in H$ is consistent with a training set of size m is $1-\delta$ and will have an error$< \epsilon$ on future examples

$$\tag{13}$$

This is called *Occam's Razor* because it expresses a preference towards smaller hypothesis spaces.
What functions are PAC Learnable?

- $|H| =$ number of conjunctions of n variables $= 3^n$ so $\ln(|H|) = n\ln(3)$
- Number of examples needed such that $m > \frac{1}{\epsilon}(\ln(|H|) - \ln(\delta)) = \frac{1}{\epsilon}(n\ln(3) - \ln(\delta))$ where δ is the confidence of finding a hypothesis space to classify the data

Not all functions are PAC Learnable though. The following is an example of a few functions which are both PAC Learnable and aren't.

- **3-CNF**: $(\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1,3}) \wedge (\ell_{2,1} \vee \ell_{2,2} \vee \ell_{2,3}) \wedge \cdots$, where each conjunct can have at most 3 literals (variables or its negatation).
  - *Sample Complexity*:
    - Number of conjuncts $= \mathcal{O}\left((2n)^3\right)$
    - A 3-CNF is a conjunction with this many variables
    - $|H| =$ number of 3-CNF's $= \mathcal{O}\left(2^{(2n)^3}\right)$
    - $\log(|H|) = \mathcal{O}\left(n^3\right)$
- **General Boolean Functions**:
  - $|H| =$ number of boolean functions exist with n variables: 2^{2^n}
  - $\log(|H|) = 2^n\log(2)$ which is exponential in n so it is **not** PAC Learnable

The following goes over a small list of functions and their sample complexity

- ***k*-CNF**: Conjunctions of any number hof clauses where each disjunctive clause has at most k literals
- ***k*-clause-CNF**: Conjunctives have at most k-disjunctive clauses

$$f = C_1 \wedge C_2 \wedge \cdots \wedge C_k$$
$$C_i = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_n$$
$$\ln(|k\text{-caluse-CNF}|) = \mathcal{O}(kn)$$

- ***k*-DNF**: Disjunctions of any number of terms wher eeach conjunctive term has at most k-literals

$$f = T_1 \vee T_2 \vee \cdots \vee T_n$$
$$T_i = \ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_k$$

- ***k*-term-DNF**: Disjunctions of at most k-conjunctive terms. $|H| = k$-CNF since a k-term-DNF can be written as a k-CNF

Agnostic Learning

- So far assumed that learning algorithms would find a concept
- What if: For a given data set, you're trying to learn a given concept f by using $h \in H$, but $f \notin H$?

With agnostic learning, you're not guarenteed that the training error will be zero.
Goal: Find a classifier $h \in H$ with a low training error

$$err_s(h) = \frac{|\{f(x) \neq h(x), x \in S\}|}{m} \tag{14}$$

which defines the fraction of traiing examples that were missclassified. We *want* a guarentee that a hypothesis with a small training error will have a good accuracy on unseen examples

$$err_{\mathcal{D}}(h) = Pr_{x \sim \mathcal{D}}(f(x) \neq h(x)) \tag{15}$$

What we need to look at are the *tail bounds*: how far a random variable can get from its mean. This can be done with a few different inequalities

- *Markov's Inequality*: Bounds the probability that a non-negative random variable exceedes a fixed value

$$P(x \geq a) \leq \frac{\langle x \rangle}{a} \tag{16}$$

- *Chebyshev's Inequality*: Bounds the probability that a random variable differs from its expected value by more than a fixed number of standard deviations

$$P(|x - \mu| \geq k\sigma) \leq \frac{1}{k^2} \tag{17}$$

  We want to bound the sums of random variables since the training error depends on the number of errors in the training set. We can do so with the following inequality

- **Hoeffding's Inequality**: Upper bounds on how much the sum of a set of random variables differs from its expected value

$$P(\mu > \bar{\mu} + \epsilon) \leq e^{-2m\epsilon^2} \tag{18}$$

  where μ is the espected mean, and $\bar{\mu}$ is the empirical mean (the mean over m trials). The empirical mean will not be too far from the expected mean if there are many samples. Hoeffding's Inequality also quantifies the convergence of this since it exponentially decays.

Applying this to Agnostic Learning
Suppose we consider the true error (generalized error) $err_{\mathcal{D}}(h)$ to be a random variable. The training error over m examples $err_s(h)$ is the empirical estimate of this true error. By Hoeffding's Inequality:

$$P(err_{\mathcal{D}}(h) > err_s(h) + \epsilon) \leq 2e^{-2m\epsilon^2} \tag{19}$$
$$err_{\mathcal{D}}(h) = Pr_{x \sim \mathcal{D}}(f(x) \neq h(x)) \tag{20}$$
$$err_s(h) = \frac{|\{f(x) \neq h(x), x \in S\}|}{m} \tag{21}$$

Therefore, the probability that a single hypothesis h has an empirical error more than ϵ away from the true error is bounded by $2e^{-2m\epsilon^2}$. Therefore, to achieve this, the learner looks for the best h possible to minimize the error. The probability that there is a hypothesis in H whose training error is ϵ away from the true error is bounded by

$$P(\exists h, err_{\mathcal{D}}(h) > err_s(h) + \epsilon) \leq |H|\, e^{-2m\epsilon^2} \tag{22}$$

we want it to be bounded by some value δ which we can make arbitrarily small

$$|H|\, e^{2m\epsilon^2} \leq \delta \tag{23}$$

$$m \geq \frac{1}{2\epsilon^2}\left(\ln(|H|) - \ln(\delta)\right) \tag{24}$$

Guarentee: Probability $1-\delta$ that the training error is not off by more than ϵ from the generalized error. We can expand this to the *generalization bound*: bound by how much the error will deviate from the training error

$$err_{\mathcal{D}}(h) - err_s(h) \leq \sqrt{\frac{\ln(|H|) - \ln(\delta)}{2m}} \tag{25}$$

- **Occam's Razor**: When the hypothesis space contains the true concept

$$m > \frac{1}{\epsilon}\left(\ln(|H|) - \ln(\delta)\right) \tag{26}$$

- **Agnostic Learning**: When the hypothesis space *may not* contain the concept

$$m > \frac{1}{2\epsilon^2}\left(\ln(|H|) - \ln(\delta)\right) \tag{27}$$

VC Dimensions and Shattering

Shattering: A set of S examples is *shattered* by a set of functions H if for every position of the examples in S into positive and negative examples, there is a function in H that gives exactly these labels to these examples.
Intuition: A rich set of functions shatters a large set of points
SHOW EXAMPLES FROM SLIDE 102 IN COLT
If an arbitrairly large finite subset of the instance space X can be shattered by a hypothesis space H, then the VC Dimension is infinite. The larger the subset X can be shattered, the more expressive a hypothesis space is, *i.e.* less biased.
VC Dimension: The VC Dimension of a hypothesis space H over instance space X is the size of the largest *finite* subset of X that is shattered by H.

- If there *exists* any subset of size d that can be shattered, then $VC(H) \geq d$
- If *no subset* of size d can be shattered, then $VC(H) < d$

Concept Class	$\boldsymbol{VC(H)}$	**Why?**
Half Intervals	1	There is a dataset of size 1 that can be shattered. No dataset of size 2 can
Intervals	2	There is a dataset of size 2 that can be shattered. No dataset of size 3 can
Half-Spaces	3	There is a dataset of size 3 that can be shattered. No dataset of size 4 can
Linear Threshold Unit	$d+1$	
Neural Networks	# Parameters	
1-Nearest Neighbors	Infinite	

We can take these results and apply them just like we did before to get a bound, such as

$$m > \frac{1}{\epsilon}\left(8VC(H)\log\left(\frac{13}{\epsilon}\right) + 4\log\left(\frac{2}{\delta}\right)\right) \tag{28}$$

with a probability $1-\delta$ has error$< \epsilon$. We can combine the VC Dimension with Agnostic Learning to get a bound on the error as well, which is

$$err_{\mathcal{D}}(h) \leq err_s(h) + \sqrt{\frac{VC(H)\ln\left(\frac{2m}{VC(H)}+1\right)+\ln\left(\frac{4}{\delta}\right)}{m}} \tag{29}$$

Boosting

Boosting is a general approach for constructing a *strong learner* given a collection of (possibly infinite) weak learners. It uses the *Ensemble Method*:

- Class of learning algorithms that composes classifiers using other classifiers as building blocks
- Boosting as strong theoretical guarentees than other ensemble methods

Goal: Automatically categorize your data based on a set of basic rules
Boosting: A general method for converting rough "rules of thumb" into accurate prediction classifiers
General Boosting Approach:

1. Select a small set of examples
2. Derive a rough "rule of thumb"
3. Select a second set of examples
4. Derive a rough "rule of thumb"
5. Repeat T times...
6. Combine all "rules of thumb" into a single prediction rule

The weighted classification error is defined as

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i) y_i h(x_i)\right) \tag{30}$$

Algorithm 1 AdaBoost($\{(\mathbf{x}_i, y_i)\}_m$)

```
Initialize $D_1(i) = \frac{1}{m}\ \forall\ i = \{1, 2, \ldots, m\}$
for $t = 1, 2, \cdots$ to $T$ do
    Find classifier $h_t$ whose weighted classificaiton error is better than chance
    Compute it's vote $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
    Update values of weights for training examples $D_{t+1} = \frac{D_t}{Z_t} \cdot \exp(-\alpha_t y_i h_t(\mathbf{x}_i))$
end for
return final hypothesis= $\text{sgn}\left(\sum_t \alpha_t h_t(\mathbf{x})\right)$
```

Why use AdaBoost:

- Simple, fast, and only one additional tuning parameter (T)
- Use with any weak learning algoirthm, only need classifiers which are slightly better than chace

Caveats:

- Performance depends on data sets and weak learners
- Can fail if weak learners are too complex (overfitting)
- Can fail is weak learners are too weak (underfitting)

Support Vector Machines

The *margin* of a hyperplane for a dataset is the distance between the hyperplane and the datapoint nearest to it. The larger the margin, the better as it generalizes the learner more and gives more "buffer room" for new examples.
Theorem Vapuik:

- Let H be the set of linear classifiers that separate the training set by a margin of at least γ, then

$$VC(H) \leq \min\left(\frac{R^2}{\gamma^2}, d\right) + 1 \tag{31}$$

$$R = \text{Radius of smallest sphere containing the data} \tag{32}$$

$$\text{Larger Margin} \Rightarrow \text{lower VC Dimension} \tag{33}$$

$$\text{Lower VC Dimension} \Rightarrow \text{Better Generalization Bound} \tag{34}$$

$$\gamma = \max_{\mathbf{w}} \min_{(\mathbf{x}_i, y_i)} \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{||\mathbf{w}||} \tag{35}$$

$$\text{SVM Prediction} = \text{sgn}(b + \mathbf{w}^T\mathbf{x}) \tag{36}$$

Max Margin Classifiers

Learning a classifier: $\min ||\mathbf{w}||$ such that the closest example is at distance $\frac{1}{||\mathbf{w}||}$. For the learning problem, we waant to minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ such that $y_i\mathbf{w}^T\mathbf{x}_i \geq 1$, which is true for eveyr example. The closest point is going to be defined as $y_i\mathbf{w}^T\mathbf{x}_i = 1$. This is known as the *Hard SVM* since it requires all of the points to be outside the margin.
The *Soft SVM* will allow some data to "break into the margin" and ignore some of the examples that make the margin smaller or inseparable. To do this we need to introduce a *slack variable* for each example, ξ_i, where we're going to require that $y_i\mathbf{w}^T\mathbf{x}_i \geq 1 - \xi_i andxi_i \geq 0\ \forall i$. This slack variable will allow some to break into the margin. If a new slack variable is zero, the example is either on or outside the margin. We now have a new optimization problem where we want to $\min_{\mathbf{w},\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$, where we want to minimize the total slack, C is the tradeoff between the two terms, and $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ is to maximize the margin. We can elminimate our slack variables and rewrite it such as

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \underbrace{\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)}_{Hinge\ Loss\ Function} \tag{37}$$

General Learning Principle: <u>Risk Minimization</u>: Define a "loss" over the training data as a function of a hypothesis. Learning is finding the hypothesis that has the lowest loss on the training data.
Regularized Risk Minimization: We define a regularzation function that penalizes over-complex hypotheses. The capacity control gives better generalization of the learner. Learning is we want to find $h(x)$ that is lowest [*regularization*+ *loss on training data*]

$$\min_{\mathbf{w}} \quad \underbrace{\frac{1}{2}\mathbf{w}^T\mathbf{w}}_{Regularization\ Term} \quad + \quad \underbrace{C}_{Hyper\ Parameter} \quad \sum_i \underbrace{\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)}_{Empirical\ Loss} \tag{38}$$

Regularization Term:

- Maximizes the margin
- Imposes a preference over the hypothesis space and pushes for better generalization
- Can be replaced with other regularization terms which impose other preferences

Empirical Loss:

- Hinges loss
- Penalizes weight vectors that make mistakes
- Can be replaces with other loss functions which impose preferences

Hyper Parameter: Controls the trade off between a large margin and a small hinge-loss

Gradient Descent

Algorithm 2 Gradient Descent for SVM

```
Initialize $\mathbf{w}^0$
for $t = 1, 2, 3, \ldots$ do
    Compute $\vec{\nabla} J(\mathbf{w}_t)$
    Update $\mathbf{w}_{t+1} = \mathbf{w}_t - r\vec{\nabla} J(\mathbf{w}_t)$
end for
return final $\mathbf{w}$
```

However, Algorithm 2 requires summing over the *entire* training set, which is really slow and doesn't scale well. An alternative that we can use is *Stochastic Gradient Descent* which treats each individual example as the entire data set. This implementation can be seen in Algorithm 3

Algorithm 3 Stoichastic Gradient Descent($S = \{(\mathbf{x}_i, y_i)\}_m$)

```
Initialize $\mathbf{w}^0$
for $epoch = 1, 2, 3, \ldots,$ to $T$ do
    Pick random example $(\mathbf{x}_i, y_i)$ from $S$
    Treat $(\mathbf{x}_i, y_i)$ as the full dataset and take the gradient for that example
        $J(\mathbf{w}_t) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$
    $\gamma_t = \frac{\gamma_0}{1+(\gamma_0 \cdot t)/C}$
    Update: $\mathbf{w}_t = \mathbf{w}_{t-1} + \gamma_t\vec{\nabla} J(\mathbf{w}_{t-1})$
end for
return final $\mathbf{w}$
```

Algorithm 3 is guarenteed to converge to the minimum of $J(\mathbf{w})$ if γ_t is small enough and $J(\mathbf{w})$ is a convex function. There is also the case of *Stoichastic sub-gradient descent*, which takes the following function

$$J(\mathbf{w}_t) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i) \tag{39}$$

Where the emperical loss function in Equation 39 is non-differentialbale. The general technique is to solve the max and then compute the gradient for each case. This technique can be seen in Algorithm 4

Algorithm 4 Stoichastic Sub Gradient Descent

```
Initialize $\mathbf{w} = \vec{0}$
for $epoch = 1, 2, 3, \ldots,$ to $T$ do
    Shuffle data
    for each training example $(\mathbf{x}_i, y_i) \in S$ do
        if $y_i\mathbf{w}^T\mathbf{x}_i \leq 1$ then
            $\mathbf{w}_t = (1 - \gamma_t)\mathbf{w}_{t-1} + \gamma_t C y_i \mathbf{x}_i$
        else
            $\mathbf{w}_t = (1 - \gamma_t)\mathbf{w}_{t-1}$
        end if
    end for
end for
return $\mathbf{w}$
```

Non-Linear Models (Support Vectors, Kernels, and Duals)

Let's say we have some vector $\mathbf{x}$ and we want to find a weight vector for it, but it's not linearly separable in 2 dimensions. We can map it to higher dimensions and predict using $\mathbf{w}^T\phi(x_1, x_2) \geq b$

$$\phi(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix} \tag{40}$$

We can make our prection via $\text{sgn}\left(\mathbf{w}^T\mathbf{x}\right)$ and $\mathbf{w}^T\mathbf{x} = \sum_i \alpha_i y_i \mathbf{x}_i^T\mathbf{x}$ meaning that we only need to compute the dot products between the training examples and a new example $\mathbf{x}_i$, even if we map it to higher dimensions. We can define a kernel $\kappa(\mathbf{x},\mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z})$, which changes to $\text{sgn}\left(\mathbf{w}^T\phi(\mathbf{x})\right) = \text{sgn}\left(\sum_i \alpha_i y_i \kappa(\mathbf{x}_i,\mathbf{x})\right)$, but is there a way to compute κ efficiently instead of doing the dot product?
Kernel Trick: You want to work with a degre 2 polynomial features, $\phi(\mathbf{x})$, then your dot product will be in space $\frac{n(n+1)}{2}$. To overcome this, a function $\kappa(\mathbf{x},\mathbf{z})$ is a valid kernel *if* it corresponds to an innder product in some (perhaps infinite dimensional) feature space.
General Condition: Construct a gram matrix $\{\kappa(\mathbf{x}_i,\mathbf{z}_j)\}$ and check to see if it's positive semi-definite. The Gram Matrix is the set of all n vectors $S = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and is the $n \times n$ matrix G, such that $G_{i,j} = \mathbf{x}_i^T\mathbf{x}_j$.
Showing a Kernel Function κ is a Valid Kernel:

- *Direct Approach*: If you have $\phi(\mathbf{x})$, you have the gram matrix and it's easy to see it's positive semi-definite
- *Indirect Approach*: If you have the kernel, write down the kernel matrix $K_{i,j}$, and show it is a legitimate kernel, without explicit construction of $\phi(\mathbf{x})$

The kernel can be shown that it's valid and positive semi-definite in the following way, which is known as *Mercer's condition.* Let $\kappa(\mathbf{x},\mathbf{z})$ be a function that maps two n-dimensional vectors to a real number. κ is a valid kernel for every set $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ for any choice of real valued $\{c_1, c_2, \ldots, c_n\}$ we have

$$\sum_j \sum_i c_i c_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{41}$$

Types of Kernels:

- *Polynomial Kernels*:
  - Linear kernels: $\kappa(\mathbf{x},\mathbf{z}) = \mathbf{x}^T\mathbf{z}$
  - Polynomial Kernels of degree d: $\kappa(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z})^d$
    * Only d^{th} order interactions
  - Polynomial kernel up to degree d: $\kappa(\mathbf{x},\mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^d,\ c > 0$
    * All interactions of order d and lower
- *Gaussian Kernel*:
  - $\kappa_{rbf}(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{||\mathbf{x}-\mathbf{z}||^2}{c}\right)$
  - $||\mathbf{x}-\mathbf{z}||^2$ = Squared euclidean distance
  - $c = \sigma^2$ = Free parameter
  - $\kappa(\mathbf{x},\mathbf{z}) \approx 1$ when $\mathbf{x}$ and $\mathbf{z}$ are close, and $\kappa(\mathbf{x},\mathbf{z}) \approx 0$ when dissimilar

Kernel Trick Example: Consider a space of 3^n monomials (positive and negative literals)

$$\kappa(\mathbf{x},\mathbf{z}) = \sum_i \phi_i^T(\mathbf{x})\phi_i(\mathbf{z}) = 2^{\text{same}(\mathbf{x},\mathbf{z})} \tag{42}$$

$$\text{same}(\mathbf{x},\mathbf{z}) = \text{number of features which are the same for } \mathbf{x} \text{ and } \mathbf{z} \tag{43}$$

Loss Minimization

Situation:

- Define a function L that penalizes bad hypotheses

- *Learning*: Pick a function $h \in H$ to minimize the expected loss

$$\min_{h \in H} E_{\mathbf{x} \sim \mathcal{D}}[L(h(\mathbf{x}), f(\mathbf{x}))] \tag{44}$$

But the distribution $\mathcal{D}$ is unknown. Instead, minimize the *empirical loss* on the training set

$$\min_{h \in H} \frac{1}{m} \sum_i L(h(\mathbf{x}_i), f(\mathbf{x}_i)) \tag{45}$$

However, we need to shy away from using complex hypotheses to avoid overfitting, which is achieved by using a *regularizer*, which penalizes complex hypotheses. This turns our minimization problem into

$$\min_{h \in H} \text{regularizer}(h) + C\frac{1}{m} \sum_i L(h(\mathbf{x}_i), f(\mathbf{x}_i)) \tag{46}$$

Loss Functions:

  – Perceptron Loss Function
$$L_{\text{perceptron}}(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\mathbf{w}^T\mathbf{x})$$
  – Hinge Loss (SVM)
$$L_{\text{hinge}}(y, \mathbf{x}, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$
  – Exponential Loss (AdaBoost)
$$L_{\text{exponential}}(y, \mathbf{x}, \mathbf{w}) = \exp(-y\mathbf{w}^T\mathbf{x})$$
  – Logistic Loss (Logistic Regression)
$$L_{\text{logistic}}(y, \mathbf{x}, \mathbf{w}) = \log(1 + \exp(-y\mathbf{w}^T\mathbf{x}))$$

Probabilistic Learning

Two different notions of probabilistic learning:

- *Learning Probabilistic Concepts*:
  – The learned concept is a function $c : X \to [0, 1]$
  – $c(x)$ may be interpreted as the probability that the label 1 is assigned to x
  – The learning theory that we have studied before is applicable (with some extensions)
- *Bayesian Learning*: Use of probabilistic criterion in selecting a hypothesis
  – The hypothesis can be deterministic, a Boolean function
  – The criterion for selecting the hypothesis is probabilistic

Bayseian Learning:
Goal: To find the *best* hypothesis from some space H of hypotheses, using the observed data $\mathcal{D}$. We define the *best* as being the *most probable hypothesis* in H. In order to do that, we need to assume a probability distribution over the class H. We also need to know something about the relation between the data and the hypothesis.
Bayes Theorem:

$$\underbrace{P(Y|X)}_{\text{Posterior}} = \frac{\overbrace{P(X|Y)}^{\text{Likelihood}}\overbrace{P(Y)}^{Prior}}{\underbrace{P(X)}_{observed}} \tag{47}$$

- *Posterior Probability*: What is the probability of Y given that X is observed?
- *Likelihood*: What is the likelihood of observing X given a specific Y?

- *Prior Probability*: What is our belief in Y before we see X?
- *Observed Probability*: What is the probability that X is observed (independent about any knowledge about the hypothesis)?

Probability Refresher:

- *Product Rule*: $P(A \wedge B) = P(A,B) = P(A|B)P(B) = P(B|A)P(A)$
- *Sum Rule*: $P(A \vee B) = P(A) + P(B) - P(A,B)$
- Events A and B are independent if:
  - $P(A,B) = P(A)P(B)$
  - Equivalently, $P(A|B) = P(A),\ P(B|A) = P(B)$
- *Theorem of Total Probability*: For mutually exclusive events $A_1, A_2, \dots, A_n$ (*i.e.* $A_i \cap A_j = \emptyset$) with $\sum_i P(A_i) = 1$

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

Choosing a Hypothesis:

- *Maximum a Posteriori* hypothesis h_{MAP}

$$h_{MAP} = \arg\max_{h \in H} P(h|D) \tag{48}$$
$$\arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)} \tag{49}$$
$$\arg\max_{h \in H} P(D|h)P(h) \tag{50}$$

- *Maximum Likelihood* hypothesis h_{ML}

$$h_{ML} = \arg\max_{h \in H} P(D|h) \tag{51}$$

  Often computationally easier to maximize the *log likelihood*

Naïve Bayes Classification: Can learn functions that predict probabilities of outcomes (different from using a probabilistic criterion to learn). *Maximum a posteriori (MAP) Prediction* as opposed to MAP Learning
MAP Prediction:

From Bayes rule for predicting y given an input x

$$P(Y = y|X = x) = \frac{P(X = x|Y = y)P(Y = y)}{P(X = x)} \tag{52}$$

predict y for the input x using

$$\arg\max_{y} \underbrace{P(X = x|Y = y)}_{Likelihood} \underbrace{P(Y = y)}_{Prior} \tag{53}$$

- *Likelihood* of observing this input x when the label is y
- *Prior* probability of the label being y

The Naïve Bayes Assumption: What if all the features were conditionally independent given the label?

$$P(x_1, x_2, \dots, x_d|y) = P(x_1|y)P(x_2|y)\cdots P(x_d|y)$$

We require only d number for each label. kd features overall.

- *Assumption*: Features are conditionally independent given the label y
- To predict, we need two sets of probabilities
  - Prior $p(y)$
  - For each x_i, we have the likelihood $p(x_i|y)$
- *Decision Rule*:

$$h_{NB}(\mathbf{x}) = \arg\max_y p(y)p(x_1, x_2, \ldots, x_d|y) \tag{54}$$

$$\arg\max_y p(y)\prod_{i=1}^{d} p(x_i|y) \tag{55}$$

We can apply this to the Maximum Likelihood Estimation which is defined as

$$h_{ML} = \arg\max_{h\in H} P(\mathcal{D}|h) \tag{56}$$

$$= \arg\max_h \prod_{i=1}^{m} p((x_i, y_i)|h) \tag{57}$$

where each example in the dataset is *independently and identically distributed*, so we can represent $p(\mathcal{D}|h)$ as this product.

$$= \arg\max_h \prod_{i=1}^{m} p(x_i|y_i, h)p(y_i|h) \tag{58}$$

$$= \arg\max_h \prod_{i=1}^{m} p(y_i|h)\prod_j p(x_{i,j}|y_i, h) \tag{59}$$

$$= \arg\max_h \sum_{i=1}^{m} \log(p(y_i|h)) + \sum_i\sum_j \log(p(x_{i,j}|y_i, h)) \tag{60}$$

For sipmlicity, suppose there are two labels $[1, 0]$ and all the features are binary. *Prior*: $p(y = 1) = p$ and $p(y = 0) = 1 - p$, while the *Likelihood* for each feature given a label is $p(x_j = 1|y = 1) = a_j$ and $p(x_j = 0|y = 1) = 1 - a_j$; $p(x_j = 1|y = 0) = b_j$ and $p(x = 0|y = 0) = 1 - b_j$. This leads to

$$p(y_i|h) = p^{[y_i=1]}(1 - p)^{[y_i=0]} \tag{61}$$

where $[z]$ is called the *indicator function* where its value is 1 if the argument z is true, and zero otherwise

$$p(x_{i,j}|y_i, h) = a_j^{[y_i=1,x_{i,j}=1]}(1 - a_j)^{[y_i=1,x_{i,j}=0]} \times b_j^{[y_i=0,x_{i,j}=1]}(1 - b_j)^{[y_i=0,x_{i,j}=0]} \tag{62}$$

Learning:

- Count how often features occur with each label. Normalize them to get likelihoods
- Priors from fraction of examples with each label
- Generalizes to multiclass

Prediction:

- Use learned probabilities to find highest scoring label

Caveats with Naïve Bayes

- Features need not be conditionally independent

$$p(x|y) \neq \prod_j p(x_j|y)$$

  Works reasonably well even when this assumption is violated

- Not enough training data to get good estimates of the probabilities from counts The basic operation for learning likelihoods is counting how often a feature occurs with a label. What if we never see a particular feature with a particular label? Should we treat those counts as zero? (that will make the probability zero) *Answer*: Smoothing:
  - Add fake counts (very small numbers so that the counts are not zero)
  - The bayesian interpretation of smoothing: *Priors* on the hypothesis (MAP Learning)

Decision Boundaries of the Naïve Bayes

Consider the two class case. We predict the label to be + if

$$p(y=+)\prod_j p(x_{i,j}|y=+) > p(y=-)\prod_j p(x_{i,j}|y=-) \tag{63}$$

$$\frac{p(y=+)\prod_j p(x_{i,j}|y=+)}{p(y=-)p(x_{i,j}|y=-)} > 1 \tag{64}$$

where we can take the log to simplify and get

$$\log\left(\frac{p(y=0|x)}{p(y=1|x)}\right) = \mathbf{w}^T\mathbf{x} + b \tag{65}$$