Support Vector Machines

Considering the linearly separable training data generated above. a) Formulate the optimization function as well as the constraints for the corresponding linear maximum margin optimization problem without a regularization term. Also show the corresponding Lagrangian as well as the Lagrangian Dual for this problem.

b) Manually perform 3 iterations of the SMO algorithm on this data. In each iteration you need to pick two α parameters, compute the unconstrained (unclipped) α values that maximize the modified performance function, clip them to make sure that both α values are ≥ 0, and then compute the corresponding decision boundary values w and b. At the end of each iteration, also provide a plot of the data points with the decision boundary described by the values of w and b that you obtained. You do not have to use any specific heuristic to pick the two α parameters in each iteration but picking them according to how likely they are to be on the margin will yield more meaningful initial decision boundaries and allow the algorithm to converge in fewer iterations

Kernel ridge regression

In contrast to ordinary least squares which has a cost function

*m*

Σ

*J*(*θ*) = 1 (*θT x*(*i*) − *y*(*i*))2, 2

*i*=1

we can also add a term that penalizes large weights in *θ*. In *ridge regression*, our least squares cost is regularized by adding a term *λ * *θ * 2, where *λ >* 0 is a fixed (known) constant (regularization will be discussed at greater length in an upcoming course lecutre). The ridge regression cost function is then

*J*(*θ*) = 1 Σ(*θT x*(*i*) − *y*(*i*))2 + *λ*

*m*

2

2

*i*=1

*θ* 2.

* 1. Use the vector notation described in class to find a closed-form expreesion for the value of *θ* which minimizes the ridge regression cost function.

Answer: Using the design matrix notation, we can rewrite *J*(*θ*) as

*J*(*θ*) = 1 (*Xθ* − *→y*)*T* (*Xθ* − *→y*) + *λθT θ*.

2 2

Then the gradient is

∇*θJ*(*θ*) = *XT Xθ* − *XT →y* + *λθ*.

Setting the gradient to 0 gives us

0 = *XT Xθ* − *XT →y* + *λθ θ* = (*XT X* + *λI*)−1*XT →y*.

* 1. Suppose that we want to use kernels to implicitly represent our feature vectors in a high-dimensional (possibly infinite dimensional) space. Using a feature mapping *φ*, the ridge regression cost function becomes

*J*(*θ*) = 1 Σ(*θT φ*(*x*(*i*)) − *y*(*i*))2 + *λ*

*m*

2

2

*i*=1

*θ* 2.

Making a prediction on a new input *x*new would now be done by computing *θT φ*(*x*new). Show how we can use the “kernel trick” to obtain a closed form for the prediction on the new input without ever explicitly computing *φ*(*x*new). You may assume that the parameter vector *θ* can be expressed as a linear combination of the input feature

vectors; i.e., *θ* = Σ*m αiφ*(*x*(*i*)) for some set of parameters *αi*.

*i*=1

[Hint: You may find the following identity useful:

(*λI* + *BA*)−1*B* = *B*(*λI* + *AB*)−1*.*

If you want, you can try to prove this as well, though this is not required for the problem.]

Answer: Let Φ be the design matrix associated with the feature vectors *φ*(*x*(*i*)). Then from parts (a) and (b),

*θ* = Φ*T* Φ + *λI* −1 Φ*T →y*

= Φ*T* ΦΦ*T* + *λI* −1 *→y*

= Φ*T* (*K* + *λI*)−1*→y.*

where *K* is the kernel matrix for the training set (since Φ*i,j* = *φ*(*x*(*i*))*T φ*(*x*(*j*)) = *Kij*.) To predict a new value *y*new, we can compute

*→y*new = *θT φ*(*x*new)

= *→yT* (*K* + *λI*)−1Φ*φ*(*x*new)

*m*

Σ

= *αiK*(*x*(*i*)*, x*new)*.*

*i*=1

where *α* = (*K* + *λI*)−1*→y*. All these terms can be efficiently computing using the kernel function.

To prove the identity from the hint, we left-multiply by *λ*(*I* + *BA*) and right-multiply by

*λ*(*I* + *AB*) on both sides. That is,

(*λI* + *BA*)−1*B* = *B*(*λI* + *AB*)−1

*B* = (*λI* + *BA*)*B*(*λI* + *AB*)−1

*B*(*λI* + *AB*) = (*λI* + *BA*)*B λB* + *BAB* = *λB* + *BAB.*

This last line clearly holds, proving the identity.

1. *ℓ*2 norm soft margin SVMs

In class, we saw that if our data is not linearly separable, then we need to modify our support vector machine algorithm by introducing an error margin that must be minimized. Specifically, the formulation we have looked at is known as the *ℓ*1 norm soft margin SVM. In this problem we will consider an alternative method, known as the *ℓ*2 norm soft margin SVM. This new algorithm is given by the following optimization problem (notice that the slack penalties are now squared):

min*w,b,ξ* 1

s.t. *y*

*w* 2 + *C* Σ*m ξ*2

(*w*

*T*

*x*(*i*) + *b*) ≥ 1 − *ξi, i* = 1*, . . . , m*

1. Notice that we have dropped the *ξi* ≥ 0 constraint in the *ℓ*2 problem. Show that these non-negativity constraints can be removed. That is, show that the optimal value of the objective will be the same whether or not these constraints are present. Answer: Consider a potential solution to the above problem with some *ξ <* 0. Then the constraint *y*(*i*)(*wT x*(*i*) + *b*) ≥ 1 − *ξi* would also be satisfied for *ξi* = 0, and the



2

(*i*)

2

*i*=1

*i*

.

objective function would be lower, proving that this could not be an optimal solution.

1. What is the Lagrangian of the *ℓ*2 soft margin SVM optimization problem?

Answer:

*m m*

L(*w, b, ξ, α*) = 1 *wT w* + *C* Σ *ξ*2 − Σ *α* [*y*(*i*)(*wT x*(*i*) + *b*) − 1 + *ξ* ],

2

where *αi* ≥ 0 for *i* = 1*, . . . , m*.

2 *i*

*i*=1

*i* *i*

*i*=1

1. Minimize the Lagrangian with respect to *w*, *b*, and *ξ* by taking the following gradients:

∇*w*L, *∂*L , and ∇*ξ*L, and then setting them equal to 0. Here, *ξ* = [*ξ*1*, ξ*2*, . . . , ξm*]*T* .

*∂b*

Answer: Taking the gradient with respect to *w*, we get

*m*

Σ

0 = ∇*w*L = *w* − *αiy*(*i*)*x*(*i*),

*i*=1

which gives us

*m*

Σ

*w* = *αiy*(*i*)*x*(*i*).

*i*=1

Taking the derivative with respect to *b*, we get

*m*

giving us

*∂b* *i*

*i*=1

0 = *∂*L = − Σ *α y*(*i*),

*m*

Σ

0 = *αiy*(*i*).

*i*=1

Finally, taking the gradient with respect to *ξ*, we have

0 = ∇*ξ*L = *Cξ* − *α*,

where *α* = [*α*1*, α*2*, . . . , αm*]*T* . Thus, for each *i* = 1*, . . . , m*, we get

0 = *Cξi* − *αi* ⇒ *Cξi* = *αi.*

1. What is the dual of the *ℓ*2 soft margin SVM optimization problem?

Answer: The objective function for the dual is

*W* (*α*) = min L(*w, b, ξ, α*)

*w,b,ξ*

*m m m*

= 1 Σ Σ(*α y*(*i*)*x*(*i*))*T* (*α y*(*j*)*x*(*j*)) + 1 Σ *αi ξ*2

Σ  ,,Σ*m* *T*  

*m*

2

*i*=1 *j*=1

*i*

*j*

2

*i*=1

*ξi*

*i*

*αi* *y*(*i*) .

*αjy*(*j*)*x*(*j*)

*x*(*i*) + *b*. − 1 + *ξi*

—

*i*=1

*j*=1

*m m m*

2

*i*

*j*

2

*i i*

= − 1 Σ Σ *α α y*(*i*)*y*(*j*)(*x*(*i*))*T x*(*j*) + 1 Σ *α ξ*

*m*

Σ

*m*

*m*

*i*=1 *j*=1

—

*i*=1

*αiy*(*i*)! *b* +

Σ*i*=1

*αi* −

Σ*i*=1

*i*=1

*αiξi*

*m m m m*

= Σ *α*

*i*

*i*=1

— 1 Σ Σ *α α y*(*i*)*y*(*j*)(*x*(*i*))*T x*(*j*) − 1 Σ *α ξ*

*i*=1 *j*=1

*i*=1

*m m m m* 2

2

*i*

*j*

2

*i i*

2

*i*

*j*

2

*C*

= Σ *α*

*i*

*i*=1

— 1 Σ Σ *α α y*(*i*)*y*(*j*)(*x*(*i*))*T x*(*j*) − 1 Σ *αi* .

*i*=1 *j*=1

*i*=1

Then the dual formulation of our problem is

*m*

*αi*

# Σ 1 Σ Σ

max*α*

*m*

*αi* −

*m*

*i*=1

2

*i*=1

(*i*)

*m*

*αiαjy*

*j*=1

2

(*j*)

*y*

(*i*) *T*

(*x*

)

(*j*) 1 Σ 2

*m i*=1

Σ

*x*

—

*i*=1 *C*

s.t. *αi* ≥ 0*, i* = 1*, . . . , m*

.

*αiy*(*i*) = 0

1. SVM with Gaussian kernel

Consider the task of training a support vector machine using the Gaussian kernel *K*(*x, z*) = exp(−*x* − *z * 2*/τ* 2). We will show that as long as there are no two identical points in the training set, we can always find a value for the bandwidth parameter *τ* such that the SVM achieves zero training error.

* 1. Recall from class that the decision function learned by the support vector machine can be written as

Σ

*m*

*f* (*x*) = *αiy*(*i*)*K*(*x*(*i*)*, x*) + *b.*

*i*=1

Assume that the training data {(*x*(1)*, y*(1))*, . . . ,* (*x*(*m*)*, y*(*m*))} consists of points which are separated by at least a distance of *ǫ*; that is, ||*x*(*j*) − *x*(*i*)|| ≥ *ǫ* for any *i* /= *j*. Find values for the set of parameters {*α*1*, . . . , αm, b*} and Gaussian kernel width *τ* such that *x*(*i*) is correctly classified, for all *i* = 1*, . . . , m*. [Hint: Let *αi* = 1 for all *i* and *b* = 0. Now notice that for *y* ∈ {−1*,* +1} the prediction on *x*(*i*) will be correct if

|*f* (*x*(*i*)) − *y*(*i*)| *<* 1, so find a value of *τ* that satisfies this inequality for all *i*.]

Answer: First we set *αi* = 1 for all *i* = 1*, . . . , m* and *b* = 0. Then, for a training example {*x*(*i*)*, y*(*i*)}, we get

*m*

*f* (*x*(*i*)) − *y*(*i*) =

*j*=1

# Σ

*m*

Σ

*y*(*j*)*K*(*x*(*j*)*, x*(*i*)) − *y*(*i*)

=

*j*=1

*y*(*j*) exp

−*x*(*j*) − *x*(*i*) 2*/τ* 2

— *y*(*i*)

= *y*(*i*) + Σ *y*(*j*) exp *x*(*j*) − *x*(*i*) 2*/τ* 2 − *y*(*i*)

*j*/=*i*

= Σ *y*(*j*) exp −*x*(*j*) − *x*(*i*) 2*/τ* 2

≤ *y*(*j*) exp −*x*(*j*) − *x*(*i*)  2*/τ* 2

Σ

*j*/=*i*

*j*/=*i*

Σ

= *y*(*j*) · exp  *x*(*j*) − *x*(*i*)  2*/τ* 2

*j*/=*i*

Σ

= exp −*x*(*j*) − *x*(*i*)  2*/τ* 2

*j*/=*i*

Σ

≤ exp −*ǫ*2*/τ* 2

*j*/=*i*

= (*m* − 1) exp −*ǫ*2*/τ* 2 *.*

The first inequality comes from repeated application of the triangle inequality |*a* + *b*| ≤

|*a*| + |*b*|, and the second inequality (1) from the assumption that ||*x*(*j*) − *x*(*i*)|| ≥ *ǫ* for all

*i* /= *j*. Thus we need to choose a *γ* such that

(*m* − 1) exp(−*ǫ*2*/τ* 2) *<* 1*,*

or *ǫ*

*τ < .*

log(*m* − 1)

By choosing, for example, *τ* = *ǫ/* log *m* we are done.

* 1. Suppose we run a SVM with slack variables using the parameter *τ* you found in part

(a). Will the resulting classifier necessarily obtain zero training error? Why or why not? A short explanation (without proof) will suffice.

Answer: The classifier will obtain zero training error. The SVM without slack variables will always return zero training error if it is able to find a solution, so all that remains to be shown is that there exists at least one feasible point.

Consider the constraint *y*(*i*)(*wT x*(*i*) + *b*) for some *i*, and let *b* = 0. Then

*y*(*i*)(*wT x*(*i*) + *b*) = *y*(*i*) · *f* (*x*(*i*)) *>* 0

since *f* (*x*(*i*)) and *y*(*i*) have the same sign, and shown above. Therefore, as we choose all the *αi*’s large enough, *y*(*i*)(*wT x*(*i*) + *b*) *>* 1, so the optimization problem is feasible.

* 1. Suppose we run the SMO algorithm to train an SVM with slack variables, under the conditions stated above, using the value of *τ* you picked in the previous part, and using some arbitrary value of *C* (which you do not know beforehand). Will this necessarily result in a classifier that achieve zero training error? Why or why not? Again, a short explanation is sufficient.

Answer: The resulting classifier will not necessarily obtain zero training error. The *C*

parameter controls the relative weights of the (*C* Σ*m ξi*) and ( 1 ||*w*||2) terms of the SVM

*i*=1

2

training objective. If the *C* parameter is sufficiently small, then the former component will have relatively little contribution to the objective. In this case, a weight vector which has a very small norm but does not achieve zero training error may achieve a lower objective value than one which achieves zero training error. For example, you can consider the extreme case where *C* = 0, and the objective is just the norm of *w*. In this case, *w* = 0 is the solution to the optimization problem regardless of the choise of *τ* , this this may not obtain zero training error.

1. Naive Bayes and SVMs for Spam Classification

In this question you’ll look into the Naive Bayes and Support Vector Machine algorithms for a spam classification problem. However, instead of implementing the algorithms your- self, you’ll use a freely available machine learning library. There are many such libraries available, with different strengths and weaknesses, but for this problem you’ll use the WEKA machine learning package, available at [http://www.cs.waikato.ac.nz/ml/weka/.](http://www.cs.waikato.ac.nz/ml/weka/) WEKA implements many standard machine learning algorithms, is written in Java, and has both a GUI and a command line interface. It is not the best library for very large-scale data sets, but it is very nice for playing around with many different algorithms on medium size problems.

You can download and install WEKA by following the instructions given on the website above. To use it from the command line, you first need to install a java runtime environ- ment, then add the weka.jar file to your CLASSPATH environment variable. Finally, you can call WEKA using the command:

java <classifier> -t <training file> -T <test file>

For example, to run the Naive Bayes classifier (using the multinomial event model) on our provided spam data set by running the command:

java weka.classifiers.bayes.NaiveBayesMultinomial -t spam train 1000.arff -T spam test.arff

The spam classification dataset in the q4/ directory was provided courtesy of Christian Shelton (cshelton@cs.ucr.edu). Each example corresponds to a particular email, and each feature correspondes to a particular word. For privacy reasons we have removed the actual words themselves from the data set, and instead label the features generically as f1, f2, etc. However, the data set is from a real spam classification task, so the results demonstrate the performance of these algorithms on a real-world problem. The q4/ directory actually con- tains several different training files, named spam train 50.arff, spam train 100.arff, etc (the “.arff” format is the default format by WEKA), each containing the corresponding number of training examples. There is also a single test set spam test.arff, which is a hold out set used for evaluating the classifier’s performance.

* 1. Run the weka.classifiers.bayes.NaiveBayesMultinomial classifier on the dataset and report the resulting error rates. Evaluate the performance of the classifier using each of the different training files (but each time using the same test file, spam test.arff). Plot the error rate of the classifier versus the number of training examples.
  2. Repeat the previous part, but using the weka.classifiers.functions.SMO classifier, which implements the SMO algorithm to train an SVM. How does the performance of the SVM compare to that of Naive Bayes?

Answer: Using the above command line arguments to run the classifier, we obtain the following error rates for the two algorithms:

8

Support Vector Machine Naive Bayes

7

6

5

Error (%)

4

3

2

1

0

0 200 400 600 800 1000 1200 1400 1600 1800 2000

Num Training Examples

For small amounts of data, Naive Bayes performs better than the Support Vector Machine. However, as the amount of data grows, the SVM achieves a better error rate.

1. Uniform convergence

In class we proved that for any finite set of hypotheses H = {*h*1*, . . . , hk*}, if we pick the

hypothesis *h*ˆ that minimizes the training error on a set of *m* examples, then with probability at least (1 − *δ*),

*ε*(*h*ˆ) ≤ min *ε*(*h* ) + 2r 1 log 2*k* ,

*i*

*i* 2*m δ*

where *ε*(*hi*) is the generalization error of hypothesis *hi*. Now consider a special case (often called the *realizable* case) where we know, a priori, that there is some hypothesis in our class H that achieves zero error on the distribution from which the data is drawn. Then we could obviously just use the above bound with min*i ε*(*hi*) = 0; however, we can prove a better bound than this.

* 1. Consider a learning algorithm which, after looking at *m* training examples, chooses

some hypothesis *h*ˆ

∈ H that makes zero mistakes on this training data. (By our

assumption, there is at least one such hypothesis, possibly more.) Show that with probability 1 − *δ*

*ε*(*h*ˆ) ≤ 1

*m*

*k*

log .

*δ*

Notice that since we do not have a square root here, this bound is much tighter. [Hint: Consider the probability that a hypothesis with generalization error greater than *γ* makes no mistakes on the training data. Instead of the Hoeffding bound, you might also find the following inequality useful: (1 − *γ*)*m* ≤ *e*−*γm*.]

Answer: Let *h* ∈ H be a hypothesis with true error greater than *γ*. Then

*P* (“*h* predicts correctly”) ≤ 1 − *γ*,

so

*P* (“*h* predicts correctly *m* times”) ≤ (1 − *γ*)*m* ≤ *e*−*γm*.

Applying the union bound,

*P* (∃*h* ∈ H*,* s.t. *ε*(*h*) *> γ* and “h predicts correct *m* times”) ≤ *ke*−*γm*.

We want to make this probability equal to *δ*, so we set

*ke*−*γm* = *δ*,

which gives us

1 *k*

*γ* = log .

*m δ*

This impiles that with probability 1 − *δ*,

*ε*(*h*ˆ) ≤ 1

*m*

*k*

log .

*δ*

* 1. Rewrite the above bound as a sample complexity bound, i.e., in the form: for fixed *δ* and *γ*, for *ε*(*h*ˆ) ≤ *γ* to hold with probability at least (1 − *δ*), it suffices that *m* ≥ *f* (*k, γ, δ*) (i.e., *f* (·) is some function of *k*, *γ*, and *δ*).

Answer: From part (a), if we take the equation,

*ke*−*γm* = *δ*

and solve for *m*, we obtain

1 *k*

*m* = log *.*

*γ δ*

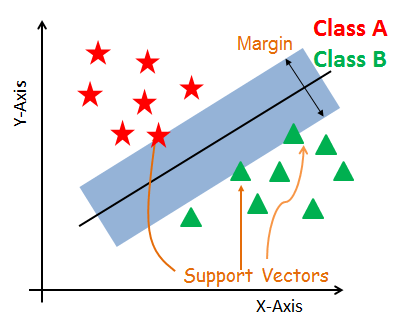
Therefore, for *m* larger than this, *ε*(*h*ˆ) ≤ *γ* will hold with probability at least 1 − *δ*.

Q2).

Consider again the problem from the previous assignments (Question 2c in Homework 1) where we want to predict the type of material of a mug based on four measurements, namely the height, diameter, weight, and hue (color). To make this into a 2 class problem, we will here consider a ”one-against-all” classification scenario where we want to predict whether the material is ”Plastic” or not (you do not have to address the other cases of the ”one-against-all” classification scenario, i.e. you do not have to learn additional binary ”Metal” or not and ”Ceramic” or not classifiers). To evaluate the learned classifier you should split the larger dataset from Homework 1, Question 2c) into a test set containing the first 6 examples of each of the materials and a training set that contains the remaining data points.

1. Use a SVM solver (e.g. MatLab’s fitcsvm function or for Python Scikit-Learn’s svm class) to learn the linear SVM parameters for this problem (linear here means that we are not using a kernel function). Since this data is not linearly separable you need to use a non-zero value for the regularization weight C (you can use the default value or experiment with different values to see the differences). Show the classification accuracy you achieved on both the test and the training set and indicate if you think the system is overfitting. Plot the data points and the resulting decision boundary’s projection in the 3-dimensional height/diameter/weight space (ignoring ”hue”). You can do this by plotting the 2-D projections of this 3D space onto the 2D subspaces height/diameter, height/weight, and diameter/weight. Also identify the support vectors in this problem.
2. Repeat the classification experiment of part a) but using the SVM with Gaussian Kernels to allow a non-linear decision boundary. In this case you have two parameters, namely the regularization weight C and the standard deviation for the Gaussian Kernels, σ. Again you can use the default values or experiment with different values to see the difference in classification accuracy and overfitting. Indicate the accuracy you achieved on the test and training set and whether you observe overfitting. Compare the results with the ones for the linear SVM and discuss your observations. Also show the classification results by plotting the data points, colored by whether they fall into the positive (”Plastic”) or negative class in the 3-dimensional height/diameter/weight space (ignoring ”hue”). You can do this again by plotting the 2-D projections of this 3D space onto the 2D subspaces height/diameter, height/weight, and ”diameter/weight” (note that since the decision boundary is highly non-linear, you do not have to plot the actual decision boundary which will be between the two colors of the points. Finally, identify the support vectors in this problem.

Generally, Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.



#### Support Vectors

Support vectors are the data points, which are closest to the hyperplane. These points will define the separating line better by calculating margins. These points are more relevant to the construction of the classifier.

#### Hyperplane

A hyperplane is a decision plane which separates between a set of objects having different class memberships.

#### Margin

A margin is a gap between the two lines on the closest class points. This is calculated as the perpendicular distance from the line to support vectors or closest points. If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

## **How does SVM work?**

The main objective is to segregate the given dataset in the best possible way. The distance between the either nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum marginal hyperplane in the following steps:

Generate hyperplanes which segregates the classes in the best way. Left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.

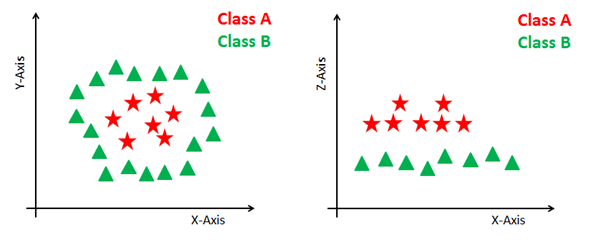
Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure.



#### Dealing with non-linear and inseparable planes

Some problems can’t be solved using linear hyperplane, as shown in the figure below (left-hand side).

In such situation, SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right. The data points are plotted on the x-axis and z-axis (Z is the squared sum of both x and y: z=x^2=y^2). Now you can easily segregate these points using linear separation.



## **SVM Kernels**

The SVM algorithm is implemented in practice using a kernel. A kernel transforms an input data space into the required form. SVM uses a technique called the kernel trick. Here, the kernel takes a low-dimensional input space and transforms it into a higher dimensional space. In other words, you can say that it converts nonseparable problem to separable problems by adding more dimension to it. It is most useful in non-linear separation problem. Kernel trick helps you to build a more accurate classifier.

* **Linear Kernel** A linear kernel can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values.

K(x, xi) = sum(x \* xi)

* **Polynomial Kernel** A polynomial kernel is a more generalized form of the linear kernel. The polynomial kernel can distinguish curved or nonlinear input space.

K(x,xi) = 1 + sum(x \* xi)^d

Where d is the degree of the polynomial. d=1 is similar to the linear transformation. The degree needs to be manually specified in the learning algorithm.

* **Radial Basis Function Kernel** The Radial basis function kernel is a popular kernel function commonly used in support vector machine classification. RBF can map an input space in infinite dimensional space.

K(x,xi) = exp(-gamma \* sum((x – xi^2))

Here gamma is a parameter, which ranges from 0 to 1. A higher value of gamma will perfectly fit the training dataset, which causes over-fitting. Gamma=0.1 is considered to be a good default value. The value of gamma needs to be manually specified in the learning algorithm.

## **Classifier Building in Scikit-learn**

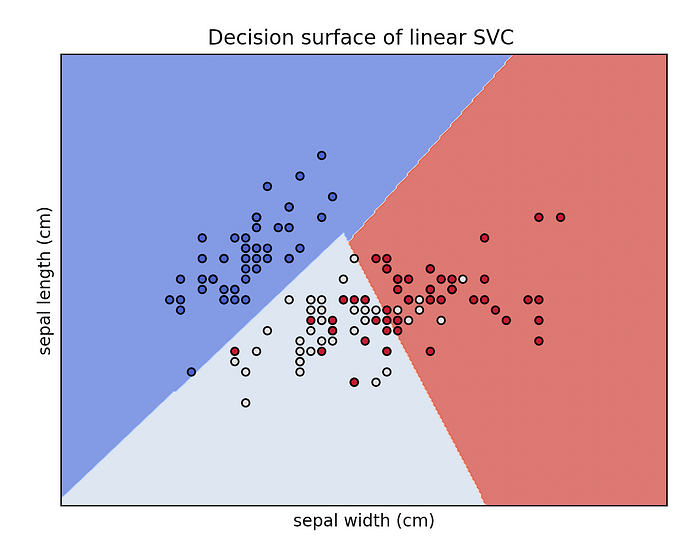
Until now, you have learned about the theoretical background of SVM. Now you will learn about its implementation in Python using scikit-learn.

In the model the building part, you can use the cancer dataset, which is a very famous multi-class classification problem. This dataset is computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

The dataset comprises 30 features (mean radius, mean texture, mean perimeter, mean area, mean smoothness, mean compactness, mean concavity, mean concave points, mean symmetry, mean fractal dimension, radius error, texture error, perimeter error, area error, smoothness error, compactness error, concavity error, concave points error, symmetry error, fractal dimension error, worst radius, worst texture, worst perimeter, worst area, worst smoothness, worst compactness, worst concavity, worst concave points, worst symmetry, and worst fractal dimension) and a target (type of cancer).

Python Code for it :-

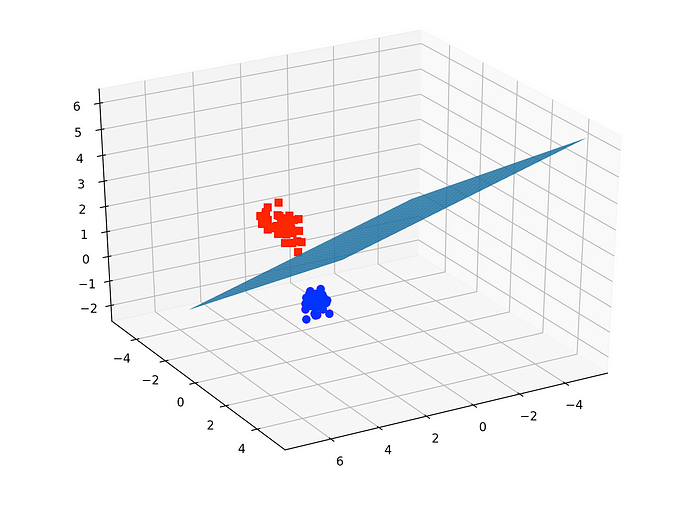
from sklearn.svm import SVC  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn import svm, datasetsiris = datasets.load\_iris()# Select 2 features / variables  
X = iris.data[:, :2] # we only take the first two features.  
y = iris.target  
feature\_names = iris.feature\_names[:2]  
classes = iris.target\_namesdef make\_meshgrid(x, y, h=.02):  
 x\_min, x\_max = x.min() — 1, x.max() + 1  
 y\_min, y\_max = y.min() — 1, y.max() + 1  
 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))  
 return xx, yydef plot\_contours(ax, clf, xx, yy, \*\*params):  
 Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])  
 Z = Z.reshape(xx.shape)  
 out = ax.contourf(xx, yy, Z, \*\*params)  
 return out# The classification SVC model  
model = svm.SVC(kernel="linear")  
clf = model.fit(X, y)fig, ax = plt.subplots()# title for the plots  
title = (‘Decision surface of linear SVC ‘)  
# Set-up grid for plotting.  
X0, X1 = X[:, 0], X[:, 1]  
xx, yy = make\_meshgrid(X0, X1)plot\_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)  
ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")  
ax.set\_ylabel("{}".format(feature\_names[0]))  
ax.set\_xlabel("{}".format(feature\_names[1]))  
ax.set\_xticks(())  
ax.set\_yticks(())  
ax.set\_title(title)  
plt.show()



# **3D**

****Let’s plot the decision boundary in 3D (we will only use 3features of the dataset):****

from sklearn.svm import SVC  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn import svm, datasets  
from mpl\_toolkits.mplot3d import Axes3Diris = datasets.load\_iris()  
X = iris.data[:, :3] # we only take the first three features.  
Y = iris.target#make it binary classification problem  
X = X[np.logical\_or(Y==0,Y==1)]  
Y = Y[np.logical\_or(Y==0,Y==1)]model = svm.SVC(kernel='linear')  
clf = model.fit(X, Y)# The equation of the separating plane is given by all x so that np.dot(svc.coef\_[0], x) + b = 0.# Solve for w3 (z)  
z = lambda x,y: (-clf.intercept\_[0]-clf.coef\_[0][0]\*x -clf.coef\_[0][1]\*y) / clf.coef\_[0][2]  
tmp = np.linspace(-5,5,30)  
x,y = np.meshgrid(tmp,tmp)fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
ax.plot3D(X[Y==0,0], X[Y==0,1], X[Y==0,2],'ob')  
ax.plot3D(X[Y==1,0], X[Y==1,1], X[Y==1,2],'sr')  
ax.plot\_surface(x, y, z(x,y))  
ax.view\_init(30, 60)  
plt.show()



Consider again the 3-

class problem from the previous assignments where we want to predict the type of material (among 3 material types) of a mug based on four measurements, namely the height, diameter, weight, and hue (color). Here we will use Decision Trees to make this prediction. Note that as the data attributes are continuous numbers yo have to use the ≤ attribute and determine a threshold for each node in the tree. As a result you need to solve the information gain for each threshold that is half way between two data points and thus the complexity of the computations increases with the number of data items.

a) Show the construction steps in the construction of a 2 level decision tree using a single step lookahead search and maximum information gain as the construction criterion. You should include the entropy calculations and the construction decisions for each node you include in the 2-level tree. Since the size of the depth-limited search used in the construction of the tree depends on the number of features and the training set size, you should limit the construction to the first 3 features (ignore ”hue”) and the data to only the first 2 data items for each material type in the data set you generated for Question 2 a) (the smaller data set for manual work) in Homework 1

### Decision Tree Algorithm

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving **regression and classification problems** too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by **learning simple decision rules** inferred from prior data(training data).

In Decision Trees, for predicting a class label for a record we start from the **root** of the tree. We compare the values of the root attribute with the record’s attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

### Types of Decision Trees

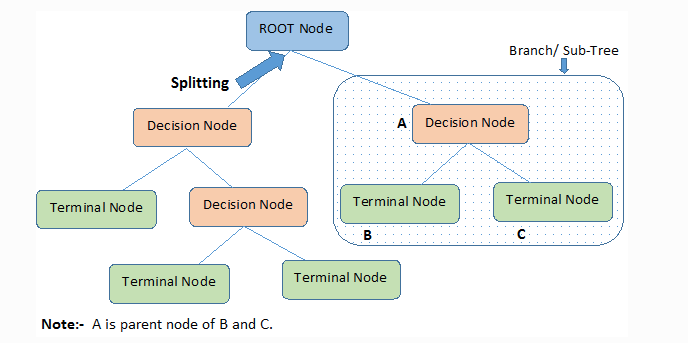
Types of decision trees are based on the type of target variable we have. It can be of two types:

1. **Categorical Variable Decision Tree:**Decision Tree which has a categorical target variable then it called a **Categorical variable decision tree.**
2. **Continuous Variable Decision Tree:**Decision Tree has a continuous target variable then it is called **Continuous Variable Decision Tree.**

**Example:-** Let’s say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that the income of customers is a significant variable but the insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree to predict customer income based on occupation, product, and various other variables. In this case, we are predicting values for the continuous variables.

### Important Terminology related to Decision Trees

1. **Root Node:**It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:**It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:**When a sub-node splits into further sub-nodes, then it is called the decision node.
4. **Leaf / Terminal Node:**Nodes do not split is called Leaf or Terminal node.
5. **Pruning:**When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
6. **Branch / Sub-Tree:**A subsection of the entire tree is called branch or sub-tree.
7. **Parent and Child Node:**A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.



Decision trees classify the examples by sorting them down the tree from the root to some leaf/terminal node, with the leaf/terminal node providing the classification of the example.

Each node in the tree acts as a test case for some attribute, and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new node.

### Assumptions while creating Decision Tree

Below are some of the assumptions we make while using Decision tree:

* In the beginning, the whole training set is considered as the **root.**
* Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
* Records are **distributed recursively** on the basis of attribute values.
* Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

Decision Trees follow **Sum of Product (SOP) r**epresentation. The Sum of product (SOP) is also known as **Disjunctive Normal Form**. For a class, every branch from the root of the tree to a leaf node having the same class is conjunction (product) of values, different branches ending in that class form a disjunction (sum).

The primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is to know as the attributes selection. We have different attributes selection measures to identify the attribute which can be considered as the root note at each level.

### How do Decision Trees work?

The decision of making strategic splits heavily affects a tree’s accuracy. The decision criteria are different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on the type of target variables. Let us look at some algorithms used in Decision Trees:

**ID3** → (extension of D3)  
**C4.5** → (successor of ID3)  
**CART** → (Classification And Regression Tree)  
**CHAID** → (Chi-square automatic interaction detection Performs multi-level splits when computing classification trees)  
**MARS** → (multivariate adaptive regression splines)

The ID3 algorithm builds decision trees using a top-down [greedy search](https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/" \t "https://www.kdnuggets.com/2020/01/_blank)approach through the space of possible branches with no backtracking. A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment.

**Steps in ID3 algorithm:**

1. It begins with the original set S as the root node.
2. On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates **Entropy(H)** and **Information gain(IG)**of this attribute.
3. It then selects the attribute which has the smallest Entropy or Largest Information gain.
4. The set S is then split by the selected attribute to produce a subset of the data.
5. The algorithm continues to recur on each subset, considering only attributes never selected before.

### Attribute Selection Measures

If the dataset consists of **N** attributes then deciding which attribute to place at the root or at different levels of the tree as internal nodes is a complicated step. By just randomly selecting any node to be the root can’t solve the issue. If we follow a random approach, it may give us bad results with low accuracy.

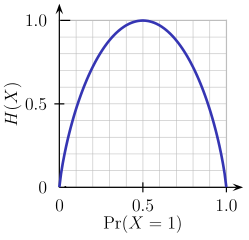
For solving this attribute selection problem, researchers worked and devised some solutions. They suggested using some criteria like :

**Entropy**,  
**Information gain,**  
**Gini index,**  
**Gain Ratio,**  
**Reduction in Variance**  
**Chi-Square**

These criteria will calculate values for every attribute. The values are sorted, and attributes are placed in the tree by following the order i.e, the attribute with a high value(in case of information gain) is placed at the root.  
While using Information Gain as a criterion, we assume attributes to be categorical, and for the Gini index, attributes are assumed to be continuous.

### ****Entropy****

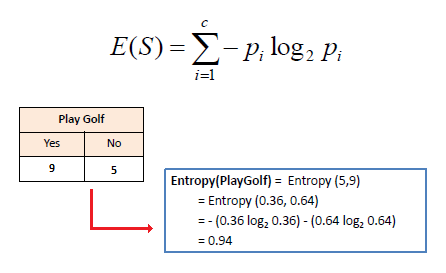
Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Flipping a coin is an example of an action that provides information that is random.



From the above graph, it is quite evident that the entropy H(X) is zero when the probability is either 0 or 1. The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data and there is no chance if perfectly determining the outcome.

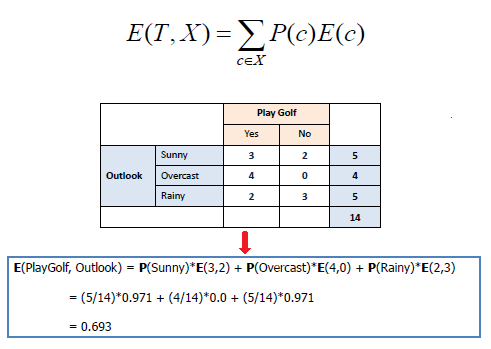
**ID3 follows the rule — A branch with an entropy of zero is a leaf node and A brach with entropy more than zero needs further splitting.**

Mathematically Entropy for 1 attribute is represented as:



Where **S → Current state, and Pi → Probability of an event**i **of state S or Percentage of class**i**in a node of state S.**

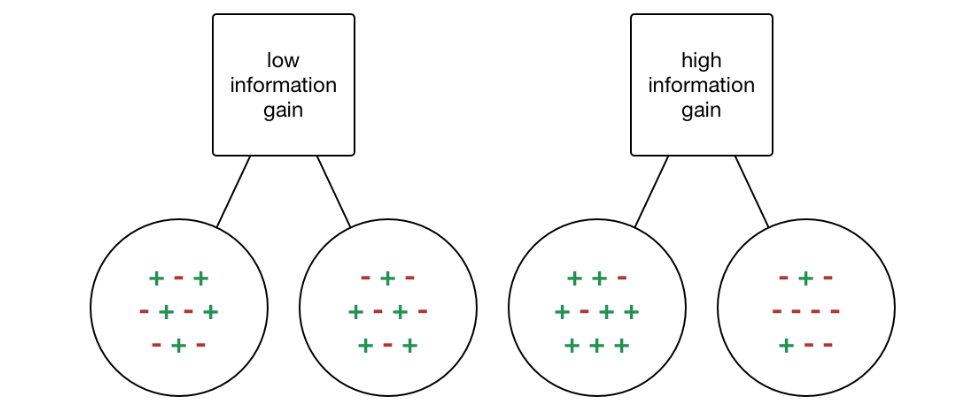
Mathematically Entropy for multiple attributes is represented as:



where**T→ Current state and X → Selected attribute**

### ****Information Gain****

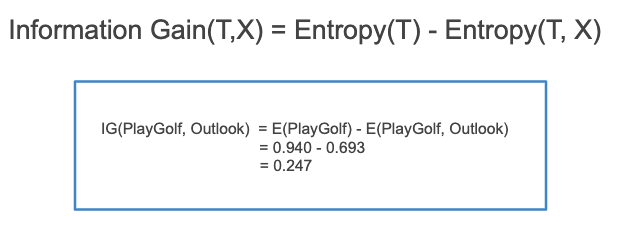
Information gain or **IG**is a statistical property that measures how well a given attribute separates the training examples according to their target classification. Constructing a decision tree is all about finding an attribute that returns the highest information gain and the smallest entropy.



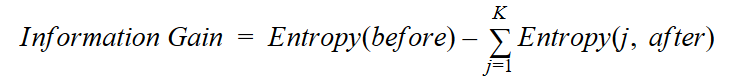
[Information Gain](https://becominghuman.ai/decision-trees-in-machine-learning-f362b296594a?gi=a8ffb5170258" \t "https://www.kdnuggets.com/2020/01/_blank)

Information gain is a decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

Mathematically, IG is represented as:



In a much simpler way, we can conclude that:

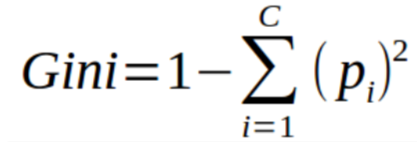


[Information Gain](https://towardsdatascience.com/from-a-single-decision-tree-to-a-random-forest-b9523be65147" \t "https://www.kdnuggets.com/2020/01/_blank)

Where “before” is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

### Gini Index

You can understand the Gini index as a cost function used to evaluate splits in the dataset. It is calculated by subtracting the sum of the squared probabilities of each class from one. It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values.



Gini Index

Gini Index works with the categorical target variable “Success” or “Failure”. It performs only Binary splits.

Higher value of Gini index implies higher inequality, higher heterogeneity.

**Steps to Calculate Gini index for a split**

1. Calculate Gini for sub-nodes, using the above formula for success(p) and failure(q) (p²+q²).
2. Calculate the Gini index for split using the weighted Gini score of each node of that split.

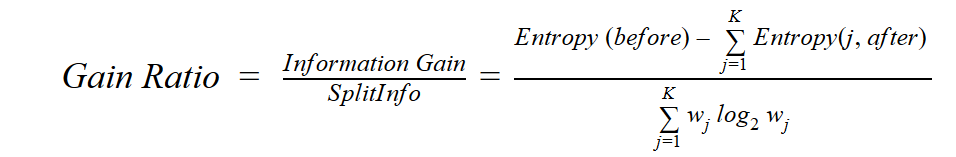
CART (Classification and Regression Tree) uses the Gini index method to create split points.

### Gain ratio

Information gain is biased towards choosing attributes with a large number of values as root nodes. It means it prefers the attribute with a large number of distinct values.

C4.5, an improvement of ID3, uses Gain ratio which is a modification of Information gain that reduces its bias and is usually the best option. Gain ratio overcomes the problem with information gain by taking into account the number of branches that would result before making the split. It corrects information gain by taking the intrinsic information of a split into account.

Let us consider if we have a dataset that has users and their movie genre preferences based on variables like gender, group of age, rating, blah, blah. With the help of information gain, you split at ‘Gender’ (assuming it has the highest information gain) and now the variables ‘Group of Age’ and ‘Rating’ could be equally important and with the help of gain ratio, it will penalize a variable with more distinct values which will help us decide the split at the next level.

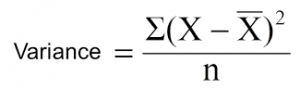


[Gain Ratio](https://towardsdatascience.com/from-a-single-decision-tree-to-a-random-forest-b9523be65147" \t "https://www.kdnuggets.com/2020/01/_blank)

Where “before” is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

### ****Reduction in Variance****

**Reduction in variance** is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with lower variance is selected as the criteria to split the population:



Above X-bar is the mean of the values, X is actual and n is the number of values.

**Steps to calculate Variance:**

1. Calculate variance for each node.
2. Calculate variance for each split as the weighted average of each node variance.

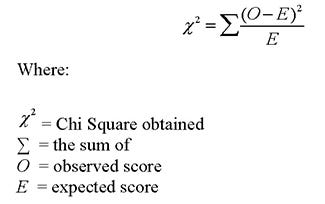
### ****Chi-Square****

The acronym CHAID stands for Chi-squared Automatic Interaction Detector. It is one of the oldest tree classification methods. It finds out the statistical significance between the differences between sub-nodes and parent node. We measure it by the sum of squares of standardized differences between observed and expected frequencies of the target variable.

It works with the categorical target variable “Success” or “Failure”. It can perform two or more splits. Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.

It generates a tree called CHAID (Chi-square Automatic Interaction Detector).

Mathematically, Chi-squared is represented as:



**Steps to Calculate Chi-square for a split:**

1. Calculate Chi-square for an individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split

### ****How to avoid/counter Overfitting in Decision Trees?****

The common problem with Decision trees, especially having a table full of columns, they fit a lot. Sometimes it looks like the tree memorized the training data set. If there is no limit set on a decision tree, it will give you 100% accuracy on the training data set because in the worse case it will end up making 1 leaf for each observation. Thus this affects the accuracy when predicting samples that are not part of the training set.

Here are two ways to remove overfitting:

1. Pruning Decision Trees.
2. Random Forest

**Pruning Decision Trees**

The splitting process results in fully grown trees until the stopping criteria are reached. But, the fully grown tree is likely to overfit the data, leading to poor accuracy on unseen data