Advanced Machine Learning Subsidary Notes

Lecture 17: Support Vector Machines

Adam Prügel-Bennett

February 9, 2024

1 Keywords

- Support Vector Machines, maximum margins

2 Main Points

2.1 Overview

- Support vector machines are one of the most successful machine learning techniques when dealing with small data sets
  - They perform binary classification (you need many SVMs to do multi-class classification)
  - There are support vector machines for regression SVR (but I'm not sure their performance dominates other techniques)
- They are basically a perceptron that regularises itself by choosing the maximum margin separating plane
- Of course perceptrons only work with linear separable data
  - SVMs overcome this limitations in two ways
    1. It often prejects the data into a high dimensional extended feature space making it more likely that the data is linear separable
       - In the extended feature space the features will usually be non-linear functions of the original features, so we can separate data in the extended feature space that isn't necessarily linear separable in the original feature space
    2. It tolerates some errors using slack variables
       - We need to tune a hyper-parameter (usually called C) to decide how much slack we want to tolerate
- SVMs can work with a huge number of features by solving the problem in the dual space
  - The dual problem depends on the number of constraints
  - There is one constraint for each pattern which states that the pattern should be on or outside a margin
  - The dual problem only depends on the inner (dot) product between vectors in the (extended) feature space
  - As positive semi-definite kernels can be represented as inner product between vectors in an extended feature space (which are the eigen-functions of the kernel) we often never need to explicitly compute the vectors in the extended feature space

2.2 Maximum Margins

- We consider data $\mathcal{D} = \{(\boldsymbol{x}_k, y_k) | k = 1, 2, \ldots, m\}$ with $\boldsymbol{x}_k \in \mathbb{R}^p$ and $y_k \in \{-1, 1\}$
- We define a dividing hyper-plane by its orthogonal vector $\boldsymbol{w}$ and a threshold b saying how far from the origin it is
- The gap between the hyper-plane and a data point $(\boldsymbol{x}_k, y_k)$ is given by

$$d_k = y_k \left(\frac{\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_k}{\|\boldsymbol{w}\|} - b \right)$$

  - Note that y_k ensures the positive class is on one side of the dividing plane and the negative class on the other
- We want to constrain these gaps to be greater than or equal to a margin Δ

$$y_k \left(\frac{\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_k}{\|\boldsymbol{w}\|} - b \right) \geq \Delta$$

- We can divide through by the margin size Δ giving

$$y_k \left(\bar{\boldsymbol{w}}^\mathsf{T} \boldsymbol{x}_k - \bar{b} \right) \geq 1$$

  - where $\bar{\boldsymbol{w}} = \boldsymbol{w} / (\Delta \, \|\boldsymbol{w}\|)$ and $\bar{b} = b/\Delta$
- We note that

$$\frac{1}{2} \|\bar{\boldsymbol{w}}\|^2 = \frac{1}{2 \, \Delta^2}$$

  - thus if we minimise $\|\bar{\boldsymbol{w}}\|^2/2$ subject to the constraint this is equivalent to maximising Δ
  - the factor of a half is just for convenience
  - we minimise $\|\bar{\boldsymbol{w}}\|^2$ rather than $\|\bar{\boldsymbol{w}}\|$ again for convenience
  - we will just drop the bar notation (i.e. we talk about $\boldsymbol{w}$ and b rather than $\bar{\boldsymbol{w}}$ and $\bar{b}$): after all what's in a name? (More seriously it has the same optimum whatever we call it)
- **Soft Margins**
  - So far we have constrained all the data points to lie outside the margin (this is known as *hard margins*)
  - We can instead allow some data points to lie within the margin (or even be misclassified)
    - ∗ This is known as a *soft margin* SVM
  - We do this by introducing slack variables s_k so that our constraints become

$$y_k \left(\frac{\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_k}{\|\boldsymbol{w}\|} - b \right) \geq 1 - s_k$$

    (I've dropped the bars for simplicity)
  - We punish the slack variables by adding $C \sum_{k=1}^{m} s_k$ to the objective function
  - We also constrain the slack variables so that $s_k \geq 0$ (so we don't get a reward if $s_k < 0$)
- Optimisation problem

- Our optimisation problem becomes

$$\min_{\boldsymbol{w},b,\boldsymbol{s}} \quad \frac{\|\boldsymbol{w}\|^2}{2} + C \sum_{k=1}^{m} s_k$$

subject to

$$y_k \left(\frac{\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_k}{\|\boldsymbol{w}\|} - b \right) \geq 1 - s_k \quad \text{and} \quad s_k \geq 0$$

for $k = 1, 2, \ldots, m$

- We can write this as a Lagrange problem

$$\min_{\boldsymbol{w},b,\boldsymbol{s}} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \quad \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

where

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\|\boldsymbol{w}\|^2}{2} + C \sum_{k=1}^{m} s_k - \sum_{k=0}^{m} \alpha_k \left(y_k \left(\boldsymbol{w}^\mathsf{T} \boldsymbol{x}_k - b \right) - 1 + s_k \right) - \sum_{k=0}^{m} \beta_k \, s_k$$

  - α_k and β_k are Lagrange multipliers
    - α_k ensures the margin condition
    - β_k ensures the non-negativity of the slack variables
  - As they enforce inequality constraints we require they are non-negative

2.3 Extended Feature Space

- To help linear separability we can map our inputs to a high-dimensional feature space

$$\boldsymbol{x} \to \boldsymbol{\phi}(\boldsymbol{x})$$

- Note in the slides I wrote this as $\vec{\phi}(\boldsymbol{x})$ to make it clear that this is a different feature space (I won't do it here as it is a slightly ugly way of writing vectors)

- Note this is a function of the original feature vector meaning that if I change $\boldsymbol{x}$ I will get a different vector $\boldsymbol{\phi}(\boldsymbol{x})$

- Nevertheless $\boldsymbol{\phi}(\boldsymbol{x})$ is just a p' dimensional vector

- We call the space of $\boldsymbol{\phi}(\boldsymbol{x})$ the *extended feature space*

- We can define this extended feature mapping explicitly (e.g. we might decide that $\phi_1(\boldsymbol{x}) = x_1^2$, $\phi_2(\boldsymbol{x}) = x_1 \, x_2$, etc.)

- More often this feature mapping is defined implicitly through a kernel function (more of that later)

- We usually choose the dimensionality, p', of the extended feature space to be much larger that that of the original feature space

- Usually we would be scared of this because it can lead to over-fitting

- However, because of choosing the maximal-margin hyper-plane this regularises the problem so strongly that we get good generalisation despite working in an enormous dimensional space

- The Lagrange problem in this extended feature space is

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{s}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\|\boldsymbol{w}\|^2}{2} + C \sum_{k=1}^{m} s_k - \sum_{k=0}^{m} \alpha_k \left(y_k \left(\boldsymbol{w}^\mathsf{T} \boldsymbol{\phi}(\boldsymbol{x}_k) - b \right) - 1 + s_k \right) - \sum_{k=0}^{m} \beta_k \, s_k$$

  - Note that the weight vector, $\boldsymbol{w}$, is now in the extended weight space

2.4 Dual Problem

- To obtain the dual problem we compute the minimum of the Lagrangian with respect to $\boldsymbol{w}$, b and $\boldsymbol{s}$
  - Weight minimisation
    $$\boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{L} = \boldsymbol{w} - \sum_{k=1}^{m} \alpha_k \, y_k \, \boldsymbol{\phi}(\boldsymbol{x}_k) = 0$$
    * So that $\boldsymbol{w} = \sum_{k=1}^{m} \alpha_k \, y_k \, \boldsymbol{\phi}(\boldsymbol{x}_k)$
  - Threshold minimisation
    $$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{k=1}^{m} \alpha_k \, y_k = 0$$
    * This gives us a global constraint on the Lagrange multipliers α_k
  - Slack variable minimisation
    $$\frac{\partial \mathcal{L}}{\partial s_k} = C - \alpha_k - \beta_k = 0$$
    * Thus $\alpha_k = C - \beta_k$
    * When substituting back into the Lagrangian all the terms proportional to s_k cancel
    * But $\beta_m \geq 0$ so this implies $\alpha_k \leq C$ (recall that $\alpha_k \geq 0$ from the KKT condition)
- Substituting the optimal values of $\boldsymbol{w}$, b and $\boldsymbol{S}$ into the Lagrangian we obtain the dual problem
  $$\max_{\boldsymbol{\alpha}} \; -\frac{1}{2} \sum_{k,l=0}^{m} \alpha_k \, \alpha_l \, y_k \, y_l \, \boldsymbol{\phi}^{\mathsf{T}}(\boldsymbol{x}_k) \, \boldsymbol{\phi}(\boldsymbol{x}_l) + \sum_{k=0}^{m} \alpha_k$$
  subject to the conditions
  $$\sum_{k=1}^{m} \alpha_k \, y_k = 0, \qquad \forall_k \; 0 \geq \alpha_k \geq C$$
  - Note that a large number of terms vanished (you really need to go through this carefully to see that this happens)
  - Amazingly adding slack variables is equivalent to preventing α_k becoming larger than C
  - the dual problem is another quadratic programming problem but involves the m Lagrange multipliers α_k rather than the p' weights
  - importantly the dual problem depends only on $\boldsymbol{\phi}(\boldsymbol{x}_k)$ through the inner product $\boldsymbol{\phi}^{\mathsf{T}}(\boldsymbol{x}_k) \, \boldsymbol{\phi}(\boldsymbol{x}_l)$

2.5 Working in the extended feature space

- If we work in the original feature space then the dual problem depends on the inner (dot) product $\boldsymbol{x}_k^{\mathsf{T}}\boldsymbol{x}_l$
  - this is known as a linear SVM
  - quite often a linear SVM gives the best performance
  - sometimes it will pay to solve the primal problem rather than dual problem
  - however we can work with very high dimensional feature vectors
    * this is useful when we work with text where out input vectors are naturally high dimensional
    * In this case we would want to solve the dual problem

- We can explicitly define our extended feature vectors
  - this has the disadvantage that they could be expensive to compute, but for some applications there may be smart ways of doing this
- **Kernel Trick**
  - There is however a really nice trick
  - Any positive definite kernel, $K(\boldsymbol{x}, \boldsymbol{y})$, can be decomposed as

    $$K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{\phi}^{\mathsf{T}}(\boldsymbol{x})\, \boldsymbol{\phi}(\boldsymbol{y})$$

    (we will talk much more about this in the next lecture)
  - obviously the functions $\phi_i(\boldsymbol{x})$ that form the elements of the vector $\boldsymbol{\phi}(\boldsymbol{x})$ depend on the kernel we choose
  - However, as the dual problem only depends on the inner product $\boldsymbol{\phi}^{\mathsf{T}}(\boldsymbol{x})\, \boldsymbol{\phi}(\boldsymbol{y})$ if we choose these so that $\boldsymbol{\phi}^{\mathsf{T}}(\boldsymbol{x})\, \boldsymbol{\phi}(\boldsymbol{y}) = K(\boldsymbol{x}, \boldsymbol{y})$ then we never have to explicitly compute $\boldsymbol{\phi}(\boldsymbol{x})$
  - Our dual problem in this case is

    $$\max_{\boldsymbol{\alpha}} \; -\frac{1}{2} \sum_{k,l=0}^{m} \alpha_k\, \alpha_l\, y_k\, y_l\, K(\boldsymbol{x}_k, \boldsymbol{x}_l) + \sum_{k=0}^{m} \alpha_k$$

  - In using an SVM we have to determine which side of the dividing plane a new data point, $\boldsymbol{x}$, would lie
    - Our prediction would be

      $$\operatorname{sgn}\big(\boldsymbol{w}^{\mathsf{T}} \boldsymbol{\phi}(\boldsymbol{x}) - b\big)$$

      - the function $\operatorname{sgn}(\cdot)$ outputs ± 1 depending on the sign of the argument
    - But our optimal weight vector is given by $\boldsymbol{w} = \sum_{k=1}^{m} \alpha_k\, y_k\, \boldsymbol{\phi}(\boldsymbol{x}_k)$
    - Thus our prediction will be

      $$\operatorname{sgn}\left(\sum_{k=1}^{m} \alpha_k\, y_k\, \boldsymbol{\phi}^{\mathsf{T}}(\boldsymbol{x}_k)\, \boldsymbol{\phi}(\boldsymbol{x}) - b \right) = \operatorname{sgn}\left(\sum_{k=1}^{m} \alpha_k\, y_k\, K(\boldsymbol{x}_k, \boldsymbol{x}) - b \right)$$

    - Thus we don't need to computer the weight or the extended feature vectors

2.6 Practical Consideration

- To get SVMs to work in practice requires a lot of tuning
- Firstly you need to normalise the inputs
- You might need to **balance** you data set
  - SVMs perform poorly on problems where the training set has many more examples of one class than the other
  - You can balance the data set by ignoring examples from the large class or
  - Increasing the size of minority class by duplicating examples
  - Sometimes you can use data augmentation on the minority class
  - This is only a problem where the data set is strongly unbalanced
- You need to choose C (this can vary by orders of magnitudes)

    - Typically found by exhaustive search start at 2^{-5} and doubling until you reach 2^{15}
    - You decide on what to use by testing on a validation set

- You also have to choose the right kernel
    - This could be no kernel (linear SVM)
    - A polynomial kernel (you have to try different degrees)
    - A radial basis kernel
    - Sometimes you use a special designed kernel for the problem
    - Kernels also come with hyper-parameters (often called γ) which you also have to tune

- It is a lot of work but for many problems (particularly with small training set) it often leads to state-of-the-art performance

- **Multi-Class Classification**
    - Natively SVMs only perform binary classification
    - If you have multiple classes you need multiple SVMs
    - People use two strategies
        - *One-versus-all*: for each class train a separate SVM using all other classes as negative examples (but see note on balancing)
        - *All-pairs*: Train a set of SVM between all pairs of classes (this is obviously expensive when you have large classes)
    - You then have to get your SVMs to vote in some way to determine the true class

3 Experiments

3.1 SVM in Practice

- SVMs are hard to code (they need a quadratic programming solver which are complicate)
- However, there a lots of good implementations out there so have a go
- Stolen from the web. . .

```
#Import scikit-learn dataset library
from sklearn import datasets

#Load dataset
cancer = datasets.load_breast_cancer()
# print the names of the 13 features
print("Features: ", cancer.feature_names)

# print the label type of cancer('malignant' 'benign')
print("Labels: ", cancer.target_names)

# print data(feature)shape
cancer.data.shape

# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test  =
          train_test_split(cancer.data, cancer.target, test_size=0.3,random_state=109)
                                  # 70% training and 30% test

#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```