OST
Ostschweizer
Fachhochschule

Lab 8

FTP MachLe MSE
HS 2024

Support Vector Machines (SVM)

MSE MachLe
WÜRC

Lernziele/Kompetenzen

- You know the primal form of a *separating hyperplane classifier*:

$$f_{w,b}(x) = \operatorname{sign}(\langle w, x \rangle + b)$$

- You can calculate the *geometric margin* γ_i of a given sample point x_i for a separating hyperplane classifier $f_{w,b}$ with given parameters w and b.

$$\gamma_i := y_i \cdot \left(\frac{\langle w, x_i \rangle + b}{\|w\|} \right)$$

- You can explain the progression of the SVM family from *maximal margin classifier* to *support vector classifier* (SVC) and finally to *support vector machines* (SVM) that use the kernel trick.
- You can use SVM successfully on tutorial style examples, including (cross-validated) parameter *grid search*.
- You know, that SVM use only a subset of training points in the decision function (called *support vectors*), so they are memory efficient. SVM are still effective in cases where number of dimensions d is greater than the number of samples N.
- You know what a *kernel function* $K(x_i, x_j)$ is and how it is related to a *feature transform* $\phi(x)$ (Mercer Theorem). Different kernel functions can be specified for the decision function of a SVM. Common kernels are the *radial basis function* kernel, the *linear* kernel and the *polynomial* kernel.

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}}$$

- You can explain the effect of the C and γ parameters for a SVM with a radial basis function kernel (rbf-kernel). If you have a lot of noisy observations, C should be small. *Decreasing C* corresponds to *higher bias*, *less variance*, *larger margin*, *higher tolerance for misclassification*, less confidence in the dataset (more noise).

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

- You know that you can always *apply the kernel trick* whenever there appears an scalar product $\langle x_i, x_j \rangle$ of two feature vectors x_i and x_j in a given loss function $\mathcal{L}$. You know that by applying the kernel trick to the dual loss function of the separating hyperplane classifier linearly non-separable classes become separable if the chosen dimension is high enough.

1. Feature Transform ϕ of the Polynomial Kernel [A,I]

The inhomogeneous quadratic polynomial kernel $K(\vec{z}_1, \vec{z}_2)$ is given by:

$$K(\vec{z}_1, \vec{z}_2) = (\vec{z}_1 \cdot \vec{z}_2 + 1)^2$$

a) Calculate the feature transform $\phi(\vec{z}) : \mathbb{R}^2 \to \mathbb{R}$ associated with the inhomogeneous quadratic polynomial kernel $K(\vec{z}_1, \vec{z}_2)$ explicitely for two-dimensional ($d = 2$) features $\vec{z} = \begin{pmatrix} x \\ y \end{pmatrix}$, i.e. find a feature transform $\phi(\vec{z})$ and an associated HILBERT space $\mathcal{H}$ such that the kernel K can be expressed as a scalar product of the transformed features $\phi(\vec{z})$.

$K(\vec{z}_1, \vec{z}_2) = (\vec{z}_1 \cdot \vec{z}_2 + 1)^2 = \langle \phi(\vec{z}_1), \phi(\vec{z}_2) \rangle_{\mathcal{H}}$

b) What is the dimensionality $\dim(\mathcal{H})$ of the associated HILBERT space, i.e. the dimensionality D of the transformed features $\phi(\vec{z})$.

c) What is the dimensionality $\dim(\mathcal{H})$ of the associated HILBERT space for the radial basis function kernel?

2. Simple SVM example [A,I]

The lecture introduced two hyper parameters of SVM: C (to penalize misclassifications) and γ (in case of the Gaussian kernel). In this exercise, you will play around with an already complete Juypter notebook to experience the effect of these parameters.

In this exercise, you will follow an experiment that tries to characterize the **cross validation (CV)** and **leave-one-out cross validation (LOOCV)** method in terms of bias and variance. This exercise further serves to make you familiar with `Jupyter` notebooks. If you haven't used `Jupyter` notebooks yet, have a look at the cheat sheets and the quick introduction given in lecture one.

a) Open the IPython Notebook `ML08_A2_SVM_Simple.jpynb` in your browser. You find it on the moodle platform.

b) Make yourself familiar with the `scikit-learn` syntax of training and applying an (SVM-) classifier in cells `In[3]` and `In[4]` as well as with the evaluation in cell `In[5]`.

c) Try some parameter values for C on your own in cell `In[5]`. What is a good range of values? What is a good strategy to search for C, with respect to sampling a certain range of possible values?
A naïve strategy could be using equidistant points, e.g.: $C = 1, 2, 3, 4, 5, 6, 7, \dots$. An often better strategy is to start with very small values and double/decuple/... the last value to use it as the next candidate. Other schemes such as *logarithmic* or *exponential* sequences are also possible.

d) Execute all cells until the end. How does the grid search parameter optimization in cell `In[6]` work?

- Does it find similar values as you did by hand?
- What strategy for sampling possible parameter values is reflected in the given values of the data structure `tuned_parameters` ?

e) Compare the CV accuracy on the training set (after cell `In[6]`) with the final scores on the test set (after cell `In[7]`).

- What is the quality of the estimated true error using CV for this particular data set?
- Which conclusions can you draw for the general case?

Further investigation [optional]: To deepen your understanding of SVM for practical applications, you may want to have a look at `scikit-learn`'s face recognition example: `https://scikit-learn.org/0.16/auto_examples/applications/face_recognition.html`. It contains code to load the *Labelled faces in the wild* data from the web, perform SVM parameter grid search, and evaluate the trained model. A complete run of the program takes approximately 5 minutes (with the most time spent on downloading the 233MB of images).

3. Predicting Diabetes using a SVM [A,II]

The data set `pima-indians-diabetes.csv` contains medical indicators for diabetes type II collected from 768 patients of the indigenous Pima tribe (Phoenix, AZ). The subjects were between 21 and 81 years old. The following characteristics were recorded:

`NumTimesPrg`	Number of pregnancies
`PlGlcConc`	Blood sugar 2h after oral uptake of sugar (oGTT) (mg/dl)
`BloodP`	Diastolic blood pressure (mm Hg)
`SkinThick`	thickness of the skin fold at the triceps (mm)
`TwoHourSerIns`	Insulin concentration after 2h oGTT (μIU/mg)
`BMI:`	Body Mass Index (kg/m^2)
`DiPedFunc`	Hereditary predisposition
`Age`	Age (years)
`HasDiabetes`	Diagnosis Diabetes Type II (target y)

The goal of the exercise is to create an optimal SVM classifier using a radial basis function kernel for the prediction of diabetes Type II. All features will be used for classification and the parameters of the radial basis function kernel (γ, C) will be tuned using a cross-validated grid search.

Open the Jupyter notebook `ML08_A3_PimaIndians_TEMPLATE.jpynb` and execute the first three cells that load the dataset into a `pandas dataframe (df)` called `pima`. The classification goal is to make the diagnosis of type II diabetes based on the factors, i.e. to make a prediction model for the variable `y=HasDiabetes` using a support vector machine (SVM) with a radial basis function kernel. Before doing this, some *data preprocessing* will be necessary.

a) Calculate the percentage of patients with diabetes and display a statistics using `pima.describe`.

b) Exclude samples with zero entries or missing values. Replace the zero values with `np.nan` and print again a statstical description of the dataset using `df.describe()`. Then drop `np.nan` values using `df.dropna()`.

c) Plot a *histogram* of the of each feature and the target using `df.hist()`.

d) *Split* the data in 80% training and 20% test data. Use `train_test_split` from `sklearn.model_sele`
If you feed a `pandas.Dataframe` as an input to the method, you will also get `pandas.Dataframes` as output for the training and test features.

e) *Standardize* the features using the `StandardScaler` from `sklearn.preprocessing` and display the histograms of the features again.

f) Train a SVM classifier with a radial basis function kernel ($\gamma = 1, C = 1$) and determine the accuracy on the test data.

g) Perform a *cross validated grid search* `GridSearchCV` to find the best parameters for γ and C and determine the best parameters and score.

- vary the value of C in the range in a logarithmic scale from 10^{-3} to 10^{+3} (7 steps)
- vary the value of γ in the range in a logarithmic scale from 10^{-3} to 10^{+3} (7 steps)
- print a classification report and a confusion matrix of the best classifier using `classification_report` and `confusion_matrix` from `sklearn.metrics`.

h) *Plot the resulting decision boundary* and margins of the SVM classifier for different values of γ and C as contour plot in 2D. Use the following two features as the only features for the prediction such that we can display the decision boundaries in a 2D plot.

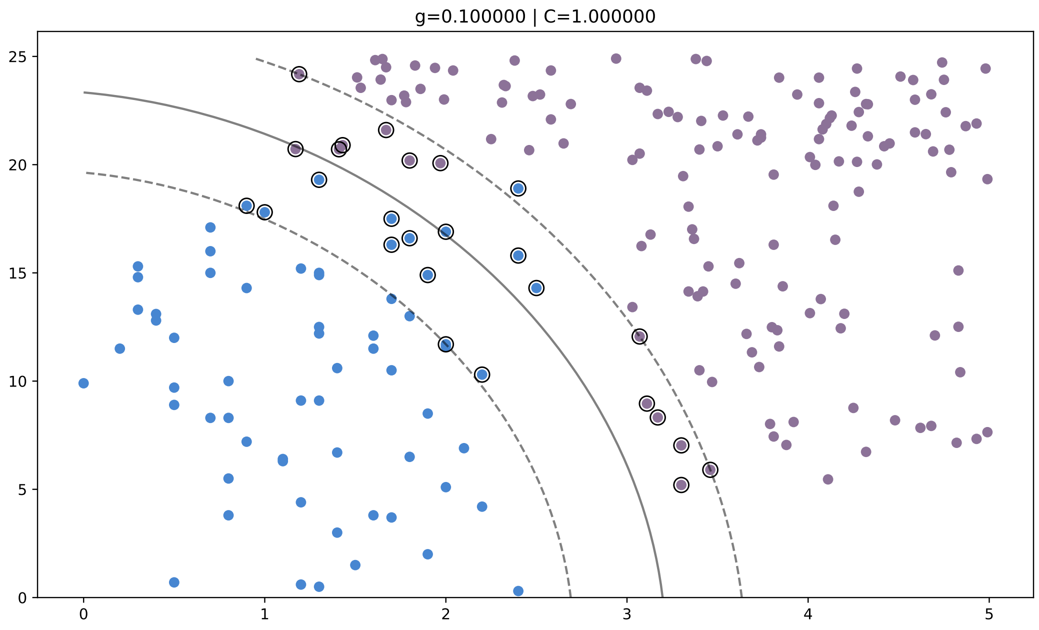
- feature 1: 'TwoHourSerIns' (x1-axis)
- feature 2: 'Age' (x2-axis)

Vary the C parameters in the following ranges: `C_range = [1e-2, 1, 1e2]` and `gamma_range = [1e-1, 1, 1e1]` and visualize the decision boundary and the margins. You can use the helper function `PlotDecisionBoundary(model, X2D, y)` which is given in the template.

4. Polynomial Kernel SVM [A,I]

Some data is almost linearly separable. The goal of this exercise is to generate nearly linearly-separable data and fit a polynomial kernel support vector machine tho the dataset and visualize the misclasifications, the decision boundary as well as the margins as function of the two parameters γ and C.

Open the IPython Notebook `ML08_A4_PolynomialKernel_TEMPLATE.jpynb` in your browser. You find it on the moodle platform. Cells 1-3 generate an almost linearly separable dataset $\mathcal{S} = \{X, y\}_{i=1...M}$ with $N = 100$.



a) Split the data in 70% training and 30% test data using `train_test_split` from `sklearn.model_selection`.

b) Plot the training data as scatter plot using different colors for both classes.

c) Fit an SVM with a second-degree polynomial kernel using $\gamma = 0.1$ and $C = 1$.

d) Plot the margins and decision boundary of the classifier.
Use the function `PlotDecisionBoundary(model,X,y)` that is provided below. The input arguments the instance of the trained `model`, the 2D array of the featues X and the 1D array of the target `y`.

```
def PlotDecisionBoundary(model, X,y):

#plot decision boundary for model in case of 2D feature space
x1=X[:,0]
x2=X[:,1]
# Create grid to evaluate model
xx = np.linspace(min(x1), max(x1), len(x1))
yy = np.linspace(min(x1), max(x2), len(x2))
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T

train_size = len(x1)

# Assigning different colors to the classes
colors = y
colors = np.where(colors == 1, '#8C7298', '#4786D1')

# Get the separating hyperplane
```

```
Z = model.decision_function(xy).reshape(XX.shape)

fig, ax = plt.subplots(figsize=(12, 7))
ax.scatter(x1, x2, c=colors)

# Draw the decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
                                  linestyles=['--', '-', '--'])

# Highlight support vectors with a circle around them
ax.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1],
                                     s=100,
linewidth=1, facecolors='none', edgecolors='k')
plt.title('g=%f | C=%f' % (model.gamma,model.C))
plt.show()
```

e) Vary the hyperparameter γ logarithmically in the range from 10^{-2} to 10^{+2} in 5 steps. Set $C = 1$ and plot for each value of the parameter γ the decision boundary and the margins.

f) Vary the hyperparameter C logarithmically in the range from 10^{0} to 10^{4} in 5 steps. Set $\gamma = 0.01$ and plot for each value of the parameter C the decision boundary and the margins.

5. General Questions about SVM [A,I]

Try to answer the following questions with 2-4 sentences.

a) What is the fundamental idea behind SVMs?

b) What is a *support vector*?

c) Why is it important to *scale* the inputs when using SVM?

d) Can an SVM output a confidence score when it classifies an instance? What about a probability?

e) Should you use the *primal* or the *dual* form of the SVM problem to train a model on a training set with millions of instances and hundreds of features?

f) Say you trained an SVM classifier with an RBF kernel. It seems to underfit the training set: should you increase or decrease γ? What about C?