DS620 Machine Learning and Deep Learning

HOS04A Support Vector Machines and Ensemble Learning

04/06/2021 Developed by Minh Nguyen

School of Technology & Computing @City University of Seattle (CityU)

Learning objectives

- Support Vector Machine Classifier
- Ensemble Learning
- Voting Classifiers
- Bagging

Resources

- scikit-learn: machine learning in Python — scikit-learn 0.24.1 documentation. (n.d.). Scikit-Learn. https://scikit-learn.org/stable/index.html
- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc.
- Bootstrap aggregating bagging. (2016, June 6). [Video]. YouTube. https://www.youtube.com/watch?v=2Mg8QD0F1dQ

Introduction

Wisdom of the crowd which refer to the concept that the collective opinion of a group of individuals is always better than that of a single expert. This often hold true when applying to Machine Learning Models, a better prediction can be made if you aggregate the predictions of a group of models, this is the basis for famous algorithms such as Decision Tree, XGBoost. In this HOS, we will cover the training process of the support vector machine algorithm and go over how

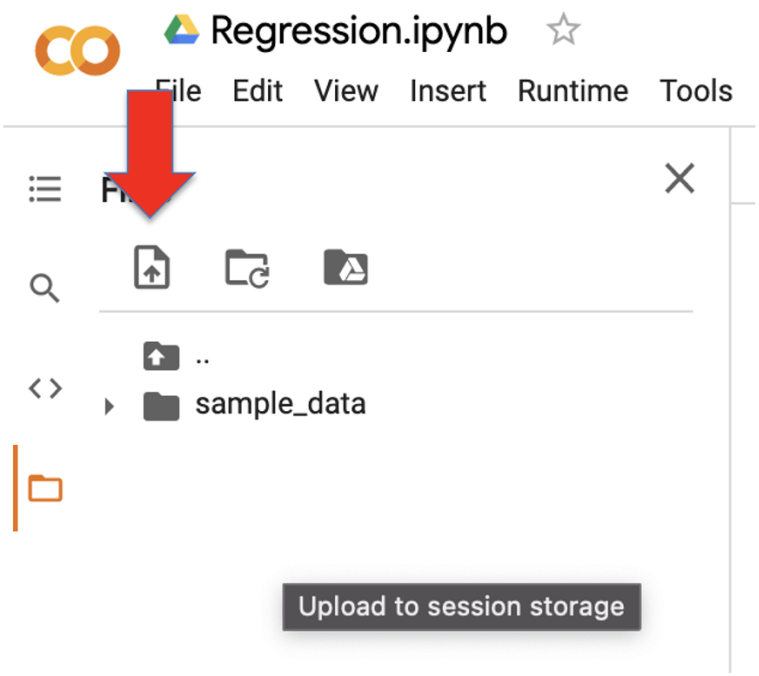
Preparing development environment

1. From Google Colab, create a new notebook, name it "SVM.ipynb"
2. Type the following codes to import libraries.

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier, BaggingClassifier
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score
```

1. Get the data

1. Upload the diabetes dataset to Google Colab



2. Import dataset

▾ Load Dataset

```
df = pd.read_csv("diabetes.csv")
df.head(10)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

3. Data Summary

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

We can see that all columns have numeric values. The "Outcome" column, which is the label column is already encoded as binary numeric data. There are 767 rows in this dataset, which is relatively low, so we should leave more sample during the splitting step.

3. Check for Missing Value

- Type the following code to check for null values

Missing values

```
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Luckily, there is no missing value in this dataset

4. Splitting data

- Splitting data set leaving 80% of the data for training.

Spliting dataset

```
# Setting random state to 620 to make the sampling reproducable
X_train, X_test, y_train, y_test = train_test_split(df.drop(columns = 'Outcome'),df['Outcome'],
                                                    train_size = 0.8, random_state = 620)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(614, 8) (614,)
(154, 8) (154,)
```

5. Preprocessing

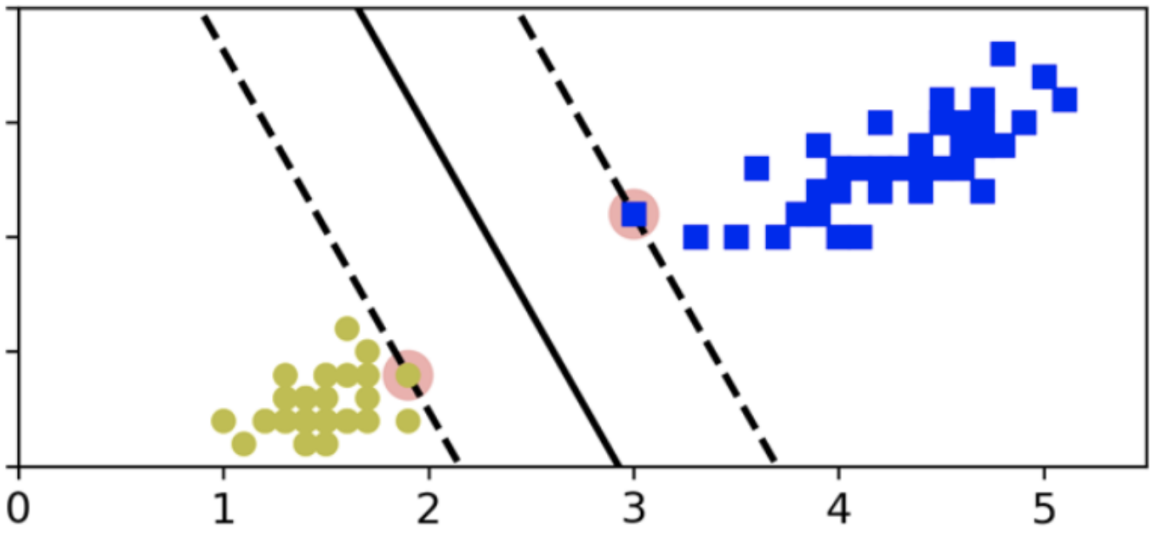
- Rescale all input features

Preprocessing

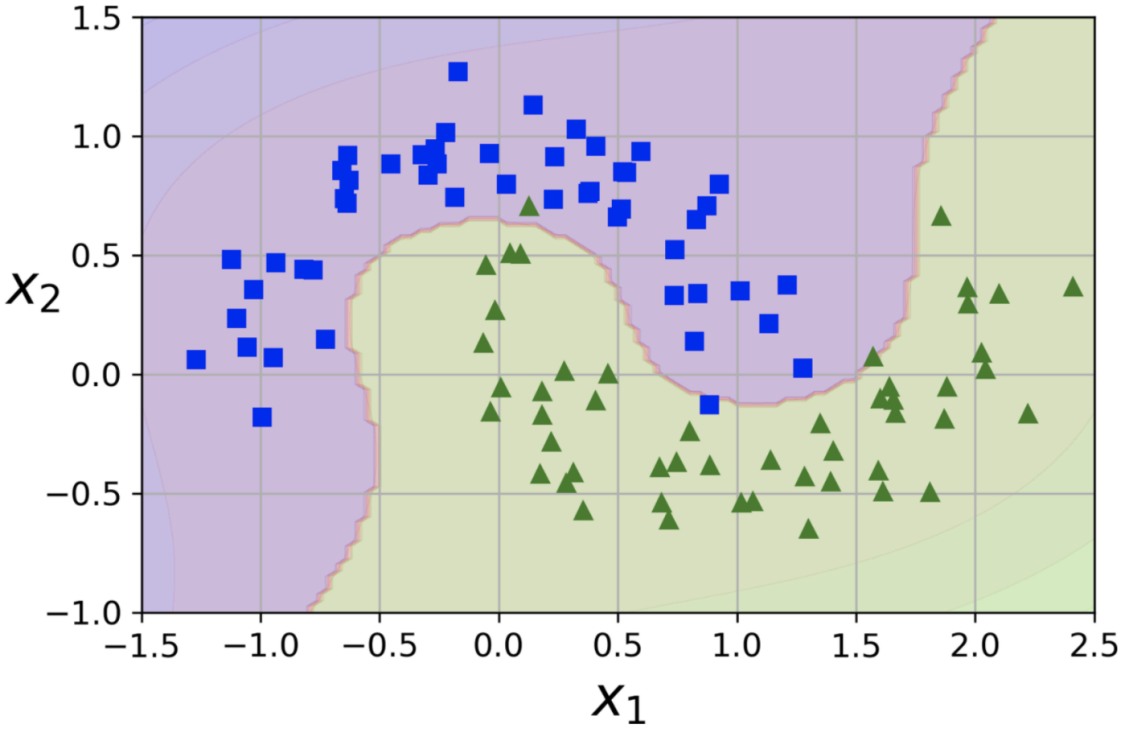
```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

6. Support Vector Machine

Support Vector Machines are perhaps one of the most popular and talked about machine learning algorithms. They were extremely popular around the time they were developed in the 1990s and continue to be the go-to method for a high-performing algorithm with little tuning. This algorithm works by separating the 2 classes of the data with a decision boundary. Here's the representation of the decision boundary in 2-dimensional space.



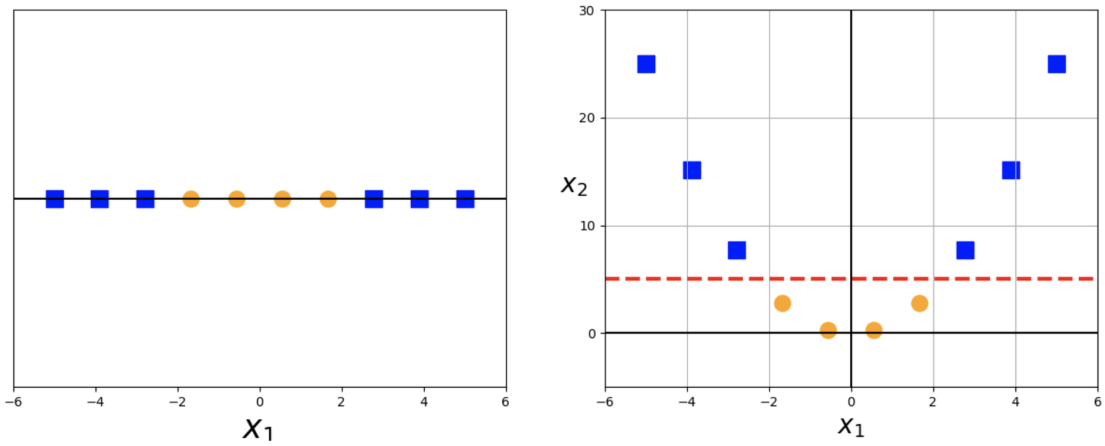
In real life the decision boundary is not always a line, it could take many forms, such as a squiggle.



In practice, The SVM algorithm is implemented using a kernel. The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. The 2 most commonly-used kernels are polynomial kernels and Radial basis function kernels.

6.1. Polynomial kernel

The polynomial works by adding polynomial interaction between the features of the input space thus adding new dimensions which make it easier to setting the decision boundary. Here's an example of transforming a 1-dimensional inputs space into 2 dimensional



Belows are the hyperparameter to keep in mind of polynomial kernels

Hyperparameter:

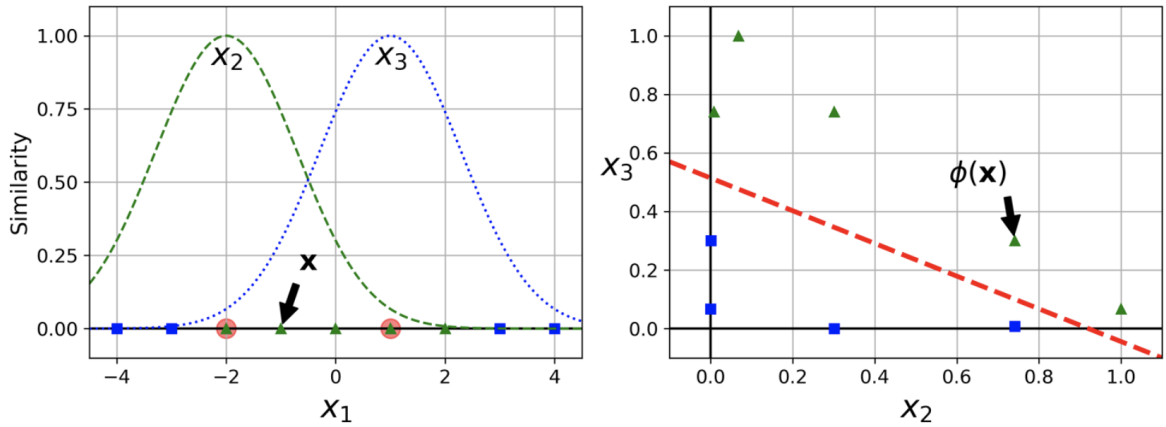
'C'**:** This determines the weight of each error. The bigger C is the less error the model is allowed to make ⇒ smaller margin. The smaller C is, the more error the model is allowed to make ⇒ Larger margin

'degree': The degree of the polynomial function, the higher the degree, the more curve there is to the data which might also lead to overfitting.

'coef0': Independent term in kernel function

6.2 Radial basis function kernel

The rbf kernel is a bell-shaped function varying from 0 (very far away from the landmark) to 1 (at the landmark).



Hyper parameter:

'C'**:** This determines the weight of each error. The bigger C is the less error the model is allowed to make ⇒ smaller margin. The smaller C is, the more error the model is allowed to make ⇒ Larger margin

'gamma': Act as the regularization parameter of the model. Large gamma overfits, small gamma under-fits

Support Vector Machine

```
[10] # Define 2 SVC models with 2 different kernels
     rbf_svc = SVC(kernel='rbf')
     poly_svc = SVC(kernel='poly')
```

```
for model in [rbf_svc,poly_svc]:
    cv_score = cross_val_score(model, X_train, y_train, scoring='accuracy' ,cv=5)
    print("Accuracy score for %s kernel SVC: %.3f"%( model.kernel, cv_score.mean()))
```

```
Accuracy score for rbf kernel SVC: 0.782
Accuracy score for poly kernel SVC: 0.735
```

+ Code + Text

Model Optimization

```
rbf_param = {'C':np.arange(0.1,1,0.1),
             'gamma':np.arange(0.01,0.1,0.01)}

rbf_tune = GridSearchCV(estimator = rbf_svc, param_grid = rbf_param,
                        scoring='accuracy',n_jobs=-1, cv=5,
                        return_train_score=True)

rbf_tune.fit(X_train,y_train)

rbf_tune.best_params_
```

```
{'C': 0.8, 'gamma': 0.060000000000000005}
```

```
poly_param = {'C':np.arange(0.5,5,0.5),
              'degree':[2,3,4,5],
              'coef0':[0,1,2]}

poly_tune = GridSearchCV(estimator = poly_svc,param_grid = poly_param,
                         scoring='accuracy', n_jobs=-1, cv=5,
                         return_train_score=True)

poly_tune.fit(X_train,y_train)

poly_tune.best_params_
```

```
{'C': 2.5, 'coef0': 1, 'degree': 3}
```

```
rbf_svc = rbf_tune.best_estimator_
poly_svc = poly_tune.best_estimator_
```

7. Voting Classifier

Voting classifier takes the prediction of multiple classifiers and choose the value with the most vote for prediction.

- Below are the code to use 3 different models for a Voting classifier.

▾ Voting Classifier

```
# Creating a voting classifier using 2 optimized SVC estimators
#along with a Decision Tree classifier
voting_clf = VotingClassifier(estimators = [('rbf_svc',rbf_svc),
                                            ('poly_svc',poly_svc),
                                            ('tree',DecisionTreeClassifier())])
```

```
# Make prediction and compare the accuracy of 3 models on new data
for model in [rbf_svc, poly_svc,DecisionTreeClassifier(), voting_clf]:
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    print("Accuracy score for %s: %.3f"%(model.__class__.__name__, accuracy_score(y_test,y_pred)))
```

```
Accuracy score for SVC: 0.740
Accuracy score for SVC: 0.721
Accuracy score for DecisionTreeClassifier: 0.734
Accuracy score for VotingClassifier: 0.747
```

The Voting classifier run 3 model simultaneously on the same dataset, get the results of the 3 models and use the one with the most vote. For example, if the both the rbf_svc and the decision tree classifier predict the class as 0, but the poly_svc predict the class as 1, the VotingClassifier will return 0 as the final result.

8. Bagging Classifier

A limitation of Voting Classifier is that all the models are trained on the same dataset which make them dependent on one another. Independency among the models ensures that they are not making the same mistakes which impossible to achieve when they are all trained on the same dataset. Bagging tackles this limitation, by creating subset of the training dataset **with replacement**, each subset is different from one another, each model will be trained on each of these subsets. Similarly, to Voting classifier, Bagging returns the prediction with the most vote. However, the difference is that Bagging Classifier requires only 1 model but it can use this model multiple times.

More on Bagging here

- Creating a Bagging Classifier by training 100 rbf SVM classification models

▾ Bagging Classifier

```
bag_svc = BaggingClassifier(base_estimator = rbf_svc, n_estimators=100,
                            max_samples=100, bootstrap=True, n_jobs=-1)
```

```
for model in [rbf_svc, bag_svc]:
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    print("Accuracy score for %s: %.3f"%(model.__class__.__name__, accuracy_score(y_test,y_pred)))
```

```
Accuracy score for SVC: 0.740
Accuracy score for BaggingClassifier: 0.760
```

Bagging slightly improve the performance of the SVM classifier

Push Your Work to GitHub

Download the notebook from Colab:

File -> SVM.ipynb

Move the downloaded file into your **Module4** working folder.

Open terminal and Navigate to the GitHub folder of this week HOS.

Make sure the assignment files on the subfolder Module4 of hos04a_YouGithubUserName folder, enter the following command to upload your work:

>>>> git add .

>>>> git commit -m “Submission for HOS04”

>>>> git push origin master