Support Vector Machines with Python

May 27, 2019

```
In [ ]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
```

0.1 Get the Data

We'll use the built in breast cancer dataset from Scikit Learn. We can get with the load function:

```
In [ ]: from sklearn.datasets import load_breast_cancer
```

```
In [ ]: cancer = load_breast_cancer()
```

The data set is presented in a dictionary form:

```
In [ ]: cancer.keys()
```

We can grab information and arrays out of this dictionary to set up our data frame and understanding of the features:

```
In [ ]: print(cancer['DESCR'])
```

```
In [ ]: cancer['feature_names']
```

0.2 Set up DataFrame

```
In [ ]: df_feat = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
        df_feat.info()
```

```
In [ ]: cancer['target']
```

```
In [ ]: df_target = pd.DataFrame(cancer['target'],columns=['Cancer'])
```

Now let's actually check out the dataframe!

```
In [ ]: df.head()
```

1 Exploratory Data Analysis

We'll skip the Data Viz part for this lecture since there are so many features that are hard to interpret if you don't have domain knowledge of cancer or tumor cells. In your project you will have more to visualize for the data.

1.1 Train Test Split

```
In [ ]: from sklearn.model_selection import train_test_split
```

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(df_feat, np.ravel(df_target), test_
```

2 Train the Support Vector Classifier

```
In [ ]: from sklearn.svm import SVC
```

```
In [ ]: model = SVC()
```

```
In [ ]: model.fit(X_train,y_train)
```

2.1 Predictions and Evaluations

Now let's predict using the trained model.

```
In [ ]: predictions = model.predict(X_test)
```

```
In [ ]: from sklearn.metrics import classification_report,confusion_matrix
```

```
In [ ]: print(confusion_matrix(y_test,predictions))
```

```
In [ ]: print(classification_report(y_test,predictions))
```

Woah! Notice that we are classifying everything into a single class! This means our model needs to have it parameters adjusted (it may also help to normalize the data).

We can search for parameters using a GridSearch!

3 Gridsearch

Finding the right parameters (like what C or gamma values to use) is a tricky task! But luckily, we can be a little lazy and just try a bunch of combinations and see what works best! This idea of creating a 'grid' of parameters and just trying out all the possible combinations is called a Gridsearch, this method is common enough that Scikit-learn has this functionality built in with GridSearchCV! The CV stands for cross-validation which is the

GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train. The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

```
In [ ]: param_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel
In [ ]: from sklearn.model_selection import GridSearchCV
```

One of the great things about GridSearchCV is that it is a meta-estimator. It takes an estimator like SVC, and creates a new estimator, that behaves exactly the same - in this case, like a classifier. You should add refit=True and choose verbose to whatever number you want, higher the number, the more verbose (verbose just means the text output describing the process).

```
In [ ]: grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3)
```

What fit does is a bit more involved then usual. First, it runs the same loop with cross-validation, to find the best parameter combination. Once it has the best combination, it runs fit again on all data passed to fit (without cross-validation), to built a single new model using the best parameter setting.

```
In [ ]: # May take awhile!
        grid.fit(X_train,y_train)
```

You can inspect the best parameters found by GridSearchCV in the best_params_ attribute, and the best estimator in the best_estimator_ attribute:

```
In [ ]: grid.best_params_
```

```
In [ ]: grid.best_estimator_
```

Then you can re-run predictions on this grid object just like you would with a normal model.

```
In [ ]: grid_predictions = grid.predict(X_test)
```

```
In [ ]: print(confusion_matrix(y_test,grid_predictions))
```

```
In [ ]: print(classification_report(y_test,grid_predictions))
```

4 Great job!