|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Support Vector Machines - Modifications | | | | | | | |
|  | **Section** | **Change** | **Previous code** | **New code** | **Description** | **Level of difficulty (1-10)** | **Time taken** |
| 1 |  | Changed style | plt.style.use('seaborn-whitegrid') | plt.style.use('seaborn-v0\_8-whitegrid') | Code gave me an error because of 'plt.style.use('seaborn-whitegrid')' part. Then I checked available styles in pyplot and identified that package called 'seaborn-whitegrid' is not in the list. Instead there was 'seaborn-v0\_8-whitegrid'. Then I replaced previous code with this new package and made the code worked. | 4 | 5 minutes |
| 2 | Motivating Support Vector Machines | Increase sample size | X, y = make\_blobs(n\_samples=50, centers=2, random\_state=0, cluster\_std=0.60) | X, y = make\_blobs(n\_samples=500, centers=2, random\_state=0, cluster\_std=0.60) | Increased sample size to 500 from 50 since this data will be used to train SVM later on this notebook. | 5 | 5 minutes |
| 3 | Fitting a Support Vector Machine | Split dataset |  | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) | In the original notebook findings has visualised without evaluating the model performance. So I decided to split the dataset to evaluate model performance before visualization. The split is 80% training and 20% testing. For this ‘train\_test\_split’ module was imported. | 3 | 5 minutes |
| 4 | Fitting a Support Vector Machine | Model Fitting using training set | model = SVC(kernel='linear', C=1E10)  model.fit(X, y) | model = SVC(kernel='linear', C=1E10)  model.fit(X\_train, y\_train)  y\_pred = model.predict(X\_test) | First created SVM classifier using Scikit-Learn's SVC (Support Vector Classification) class. kernel='linear' suggested that SVM should use a linear kernel. Then fitted the model only using training set and predicted using X\_test | 7 | 10 minutes |
| 5 | Fitting a Support Vector Machine | Evaluate model |  | print("Classification Report:\n", classification\_report(y\_test, y\_pred))  print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred)) | Model was tested using test set. Confusion matrix and classification report was obtained to evaluate the fitted model. Both suggested that model is accurate. | 5 | 10 minutes |
| 6 | Fitting a Support Vector Machine | modified 'plot\_svc\_decision\_function' | def plot\_svc\_decision\_function(model, ax=None, plot\_support=True): | def plot\_svc\_decision\_function(model, X\_train, y\_train, ax=None, plot\_support=True): | Modified 'plot\_svc\_decision\_function' in a way that consider given X, y data values. Then we can plot all dataset or only training set according to the requirement. | 8 | 20 minutes |
| 7 | Beyond Linear Boundaries: Kernel SVM | Increasing sample size | X, y = make\_circles(100, factor=.1, noise=.1) | X, y = make\_circles(500, factor=.1, noise=.1) | Increased sample size to 500 from 100 since I planned to perform data split and model training in this section. | 5 | 5 Minutes |
| 8 | Beyond Linear Boundaries: Kernel SVM(Rbf kernel) | Data split and model train | clf = SVC(kernel='rbf', C=1E6)  clf.fit(X, y) | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  clf = SVC(kernel='rbf', C=1E6, gamma='scale')  clf.fit(X\_train, y\_train) | Split the data for better model evaluation. Fit the model only using training dataset | 3 | 5 Minutes |
| 9 | Beyond Linear Boundaries: Kernel SVM(Rbf kernel) | Evaluate SVM model |  | y\_pred = clf.predict(X\_test)  print("Test Accuracy:", accuracy\_score(y\_test, y\_pred))  print("Classification Report:\n", classification\_report(y\_test, y\_pred)) | Test the model using testing data and obtained accuracy score and classification report to check the model performance. Since the model is accurate visualize the plot using all 500 data points instead of training set. | 5 | 10 Minutes |
| 11 | Tuning the SVM: Softening Margins | Find best c and gamma ‘rbf’ model |  | param\_grid = {  'C': [0.1, 1, 10, 100, 1000, 1E6],  'gamma': [0.01, 0.1, 1, 10, 'scale', 'auto']  }  clf = SVC(kernel='rbf')  grid\_search = GridSearchCV(clf, param\_grid, cv=5, scoring='accuracy', n\_jobs=-1)  grid\_search.fit(X\_train, y\_train) | Existing notebook has not evaluated tuning margins for ‘rbf’ models. Used cv=5 grid search to find best c value and gamma. For this ‘GridSearchCV’ package was imported. First defined the parameter grid and created an SVM Classifier with an RBF Kernel. Then initiated grid search and trained multiple SVM models with different hyperparameters from ‘param\_grid’ | 9 | 15 minutes |
| 12 | Tuning the SVM: Softening Margins | Train and test SVM with best c and gamma |  | best\_clf = SVC(kernel='rbf', C=0.1, gamma=1)  best\_clf.fit(X\_train, y\_train)  y\_pred = best\_clf.predict(X\_test)  print("Classification Report:\n", classification\_report(y\_test, y\_pred)) | From above step found that c=0.1 and gamma=1 are the best values for SVM for this case. So the best model was fitted using those parameters and tested with the test dataset. Then classification report was obtained to check the model performance. | 7 | 15 minutes |
| 13 | Tuning the SVM: Softening Margins | Visualise the best model |  | plt.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap='autumn', s=50)  plot\_svc\_decision\_function(best\_clf, X, y) | Since the model is performing well, visualisation is made using ‘plot\_svc\_decision\_function’. | 7 | 10 minutes |
| 14 | Example: Face Recognition | Normalize pixels and flatten images |  | X = lfw\_people.images / 255.0  X = X.reshape(X.shape[0], -1) | Pixel normalization and reshaping images to 2D was done as a preprocessing step before PCA. This ensures that features are balanced, and model is generalized. | 7 | 10 minutes |
| 15 | Example: Face Recognition | Evaluate model with ‘min\_faces\_per\_person=60’ |  | print(classification\_report(ytest, yfit,  target\_names=faces.target\_names)) | Face recognition model was trained using ‘min\_faces\_per\_person=60’ condition. After testing this model, I obtained classification report to check the model accuracy and found that accuracy is 85%.  Then I planned to do this same model training with ‘min\_faces\_per\_person=100’ condition | 5 | 5 minutes |
| 16 | Example: Face Recognition | Increase ‘ min\_faces\_per\_person’ |  | faces = fetch\_lfw\_people(min\_faces\_per\_person=100) | I Increased ‘min\_faces\_per\_person’ to 100 and model was trained and tested using the same procedure above. Then the model accuracy became 91% as I was using more data for the model trainning. | 7 | 20 minutes |

**My Example**

**Application of SVM - Iris dataset**

The Iris dataset consists of flower measurements (sepal length, sepal width, petal length, petal width) and their corresponding species (Setosa, Versicolor, or Virginica). This model tries to classify iris flowers into the correct species using Support Vector Machines (SVM) classifier.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Purpose | Code | Description | Level of difficulty (1-10) | Time spent |
| 1 | Importing Libraries | from sklearn.datasets import load\_iris | For this ‘load\_iris’ was imported from ‘sklearn.datasets’. All libraries have included at the beginning of the notebook to avoid redundancy. | 3 | 5 minutes |
| 2 | Loading dataset and assigning X,y | iris = datasets.load\_iris()  X = iris.data  y = iris.target | In this step dataset has loaded. Then split into data and target. X stores data and y stores target values. | 5 | 10 minutes |
| 3 | Train -test split | X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.4, random\_state=42, stratify=y)  X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42, stratify=y\_temp) | Dataset has been split into 3 parts. 60% training (X\_train, y\_train), 20% validation (X\_val, y\_val) and 20% testing (X\_test, y\_test). Here ‘random\_state=42’ used to ensure the reproducibility of the datasets in each execution. Stratify=y parameter ensures that the train and test datasets have the same class distribution as the original dataset. | 5 | 5 minutes |
| 4 | Feature scaling | scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_val = scaler.fit\_transform(X\_val)  X\_test = scaler.transform(X\_test) | Normalized data using standard scaler. In SVM it is essential as it uses distances between data points to find the optimal decision boundary. For this ‘StandardScaler’ was imported from ‘sklearn.preprocessing’ | 6 | 10 minutes |
| 5 | Hyperparameter tuning | param\_grid = {  'C': [0.1, 1, 10, 100],  'kernel': ['linear', 'rbf', 'poly'],  'gamma': ['scale', 'auto']  }  grid\_search = GridSearchCV(SVC(), param\_grid, cv=5, scoring='accuracy', n\_jobs=-1)  grid\_search.fit(X\_train, y\_train) | This step I performed hyperparameter tuning for SVM classifier using Grid Search with 5-fold cross-validation. First, I defined the parameter grid and initiate the grid search. Then I defined 5-fold cv here which splits the provided dataset into 5 subsets and train the gridsearch using 4 subsets among them. This repeats 5 times changing data subsets. After defining this, gridsearch has been trained and evaluated in each combination of the parameter grid and provides the best parameters to use for the model. | 8 | 15 minutes |
| 6 | Training the model with optimal parameters | best\_model = grid\_search.best\_estimator\_  print(f"Best Parameters: {grid\_search.best\_params\_}")  y\_val\_pred = best\_model.predict(X\_val) | After getting optimal parameters using gridsearch method, SVM classification model can be trained again with optimal parameters using X\_train and y\_train. Then predicted the model using validation set. | 7 | 10 minutes |
| 7 | Evaluate performance | accuracy = accuracy\_score(y\_val, y\_val\_pred)  print(f"\nModel Accuracy: {accuracy:.4f}")  print("\nClassification Report:\n", classification\_report(y\_val, y\_val\_pred))  conf\_matrix = confusion\_matrix(y\_val, y\_val\_pred)  print("\nConfusion Matrix:\n", conf\_matrix) | To evaluate the model, accuracy score, classification report and confusion matrix have been conducted considering validation set. Since the model performing well according to these parameters I planned predict the model for the testing dataset. | 4 | 5 minutes |
| 8 | Final testing | y\_pred = best\_model.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  print(f"\nModel Accuracy: {accuracy:.4f}")  print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))  conf\_matrix = confusion\_matrix(y\_test, y\_pred)  print("\nConfusion Matrix:\n", conf\_matrix) | Model was predicted on testing dataset. Obtained accuracy score, classification report and confusion matrix to evaluate the model performance. Here accuracy score is 93% which is good. | 5 | 10 minutes |