|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Naive Bayes Classification - Modifications | | | | | | | |
|  | **Section** | **Change** | **Previous code** | **New code** | **Description** | **Level of difficulty (1-10)** | **Time taken** |
| 1 | Gaussian Naive Bayes | Plot probability distributions |  | sns.histplot(yprob[:, 1], bins=30, kde=True, color='blue', label="Probability of Class 1")  sns.histplot(yprob[:, 0], bins=30, kde=True, color='red', label="Probability of Class 0") | Visualised probability distribution obtained from 'yprob = model.predict\_proba(Xnew)' for better understanding and model evaluation | 9 | 20 minutes |
| 2 | Gaussian Naive Bayes | Split dataset |  | X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.4, random\_state=42)  X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42) | Previously model was evaluated by generating new dataset (Xnew). But sometimes it may not work well. So here I performed data split on same X,y which had created at the beginning of this notebook. Split was 60% training and 20% validating and 20% testing. Here ‘random\_state=42’ used to ensure the reproducibility of the datasets in each execution. Before this 'train\_test\_split' package was imported | 4 | 5 minutes |
| 3 | Gaussian Naive Bayes | Fitting the model and testing |  | model = GaussianNB()  model.fit(X\_train, y\_train)  y\_val\_pred = model.predict(X\_val) | Fitted the model using training dataset and predicted the model with X\_val set | 5 | 5 minutes |
| 4 | Gaussian Naive Bayes | Confusion matrix |  | cm = confusion\_matrix(y\_val\_pred, y\_val)  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])  plt.xlabel("Predicted Label")  plt.ylabel("True Label")  plt.title("Confusion Matrix") | This step was performed to evaluate the model performance. According to this matrix model was performing great. | 5 | 10 minutes |
| 5 | Gaussian Naive Bayes | Testing the model |  | y\_pred = model.predict(X\_test)  cm = confusion\_matrix(y\_test, y\_pred)  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])  plt.xlabel("Predicted Label")  plt.ylabel("True Label")  plt.title("Confusion Matrix") | Since validation set is performing well with the model test set also predicted and evaluated. This is more generalized way of fitting model rather than generating new dataset for the model evaluation | 6 | 10 minutes |
| 6 | Multinomial Naive Bayes | Accuracy score |  | print("Accuracy:", accuracy\_score(test.target, labels)) | In the original notebook, only the confusion matrix was obtained. I obtained the accuracy score here to compare performance after improving the pipeline. Here accuracy was 80% | 4 | 5 minutes |
| 7 | Multinomial Naive Bayes | Improve the pipeline | model = make\_pipeline(TfidfVectorizer(), MultinomialNB()) | model1 = make\_pipeline(TfidfVectorizer(stop\_words='english', ngram\_range=(1,2), max\_features=5000),MultinomialNB()) | Few additional parts were added here to improve query performance. stop\_words='english' to remove common English words like "the," "is," "and,". ngram\_range=(1,2) to consider both single words (unigrams) and two-word combinations (bigrams) | 8 | 15 minutes |
| 8 | Multinomial Naive Bayes | Fit and evaluate the model |  | model1.fit(train.data, train.target)  labels1 = model1.predict(test.data)  cm = confusion\_matrix(test.target, labels1)  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabels=set(train.target\_names), yticklabels=set(train.target\_names))  accuracy\_score(test.target, labels1) | Fitted and evaluated the model after improving the pipeline. Then confusion matrix and accuracy score were obtained. Accuracy has improved to 88% by improving the pipeline. | 6 | 15 minutes |
| 9 | Multinomial Naive Bayes | Hyperparameter Tuning |  | param\_grid = {'multinomialnb\_\_alpha': [0.01, 0.1, 1, 10]}  grid = GridSearchCV(model1, param\_grid, cv=5)  grid.fit(train.data, train.target) | Performed hyperparameter tuning to find the best settings for the model. For this 'GridSearchCV' was imported. From these steps I found out best alpha value for the model which is 0.1 | 7 | 15 minutes |
| 10 | Multinomial Naive Bayes | Train the final model |  | best\_model = make\_pipeline(TfidfVectorizer(stop\_words='english', ngram\_range=(1,2), max\_features=5000),MultinomialNB(alpha=0.1))  best\_model.fit(train.data, train.target)  predictions = best\_model.predict(test.data) | After finding the best alpha value final model was trained and evaluated. | 6 | 10 minutes |
| 11 | Multinomial Naive Bayes | Model accuracy |  | print("Accuracy:", accuracy\_score(test.target, predictions)) | Checked the model accuracy and found out that the model is 91% accurate. For this 'accuracy\_score' package was imported. With previous pipeline model accuracy was 80%. So modifiying pipeline and finding best alpha has improved the model performance. | 4 | 5 minutes |
| 12 | Multinomial Naive Bayes | Confusion matrix on final model |  | cm = confusion\_matrix(test.target, predictions)  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False, xticklabels=set(train.target\_names), yticklabels=set(train.target\_names)) | From this we can evaluate the performance of the best model | 4 | 5 minutes |
| 13 | Multinomial Naive Bayes | Testing with the best model |  | def predict\_category\_best(s, train=train, model=best\_model):  pred\_best = best\_model.predict([s])  return train.target\_names[pred\_best[0]] | Modified the function to get values from the best model trained using alpha 0.1 | 5 | 10 minutes |
| 14 | Multinomial Naive Bayes | Testing with probabilities |  | def predict\_category\_with\_prob(s, train=train, model=best\_model):  pred = best\_model.predict([s])[0]  prob = best\_model.predict\_proba([s])[0]  category = train.target\_names[pred] | Modified the function to get the output with probabilities. This will help to understand the model better | 8 | 20 minutes |

**My Example**

**Application of Gaussian Naive Bayes - Breast Cancer dataset**

Breast cancer dataset is a python default dataset which has around 30 features. This model is trained to identify whether it is a malignant cancer (1) or benign cancer (0) based on features extracted from cell nuclei measurements.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Purpose | Code | Description | Level of difficulty (1-10) | Time spent |
| 1 | Importing Libraries | from sklearn.datasets import load\_iris | For this ‘load\_breast\_cancer’ was imported from ‘sklearn.datasets’. All libraries have included at the beginning of the notebook to avoid redundancy. | 3 | 5 minutes |
| 2 | Loading the dataset, split into features, target and make a data frame | data = load\_breast\_cancer()  X, y = data.data, data.target  df = pd.DataFrame(X, columns=data.feature\_names)  df['target'] = y | In this step dataset has loaded and split into data and target. Then the dataset was converted to a data frame to check for the missing data and class distribution easily. | 5 | 10 minutes |
| 3 | Missing values and data distribution | print("Missing Values:\n", df.isnull().sum())  print("Class Distribution:\n", df['target'].value\_counts()) | Missing values and distribution of the dataset were observed using these queries. | 3 | 5 minutes |
| 4 | Splitting the dataset | X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.4, random\_state=42)  X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42) | 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. | 3 | 5 minutes |
| 5 | Initialize and train the Naïve Bayes model | model = GaussianNB()  model.fit(X\_train, y\_train)  y\_val\_pred = model.predict(X\_val) | Then initialised Gaussian NB and trained the model using X\_train and y\_train. Then predicted it using X\_val dataset. | 5 | 5 minutes |
| 6 | Evaluate the model | accuracy = accuracy\_score(y\_val, y\_val\_pred)  print("Accuracy:", accuracy)  print("Classification Report:\n", classification\_report(y\_val, y\_val\_pred))  print("Confusion Matrix:\n", confusion\_matrix(y\_val, y\_val\_pred)) | Model was evaluated considering y\_val and y\_val\_pred. Classification report, accuracy score and confusion matrix were considered when the model is evaluating. Model accuracy was 97.36% for this basic model. This high accuracy might be a hint of overfitting. So I decided to perform feature selection for this model. | 5 | 10 minutes |
| 7 | Feature Selection using SelectKBest (Chi-Square Test) | selector = SelectKBest(chi2, k=10)  X\_selected = selector.fit\_transform(X, y)  selected\_features = np.array(data.feature\_names)[selector.get\_support()]  print("Selected Features:", selected\_features) | This breast cancer dataset has 30 features. All of them might not affect to the target. So, in this step I selected the most influential 10 features using Chi-Square Test. From this I identified 'mean radius', 'mean texture', 'mean perimeter', 'mean area', 'perimeter error', 'area error', 'worst radius', 'worst texture', 'worst perimeter' and 'worst area' are the 10 most influential factors for the response. | 9 | 20 minutes |
| 8 | Improved Model (With Feature Selection) | X\_train\_sel, X\_temp, y\_train\_sel, y\_temp = train\_test\_split(X\_selected, y, test\_size=0.4, random\_state=42)  X\_val\_sel, X\_test\_sel, y\_val\_sel, y\_test\_sel = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)  model\_sel = GaussianNB()  model\_sel.fit(X\_train\_sel, y\_train\_sel)  y\_val\_pred\_sel = model\_sel.predict(X\_val\_sel)  accuracy\_sel = accuracy\_score(y\_val\_sel, y\_val\_pred\_sel)  print("Accuracy (Feature Selected Model):", accuracy\_sel)  print("Classification Report (Feature Selected Model):\n", classification\_report(y\_val\_sel, y\_val\_pred\_sel))  print("Confusion Matrix (Feature Selected Model):\n", confusion\_matrix(y\_val\_sel, y\_val\_pred\_sel)) | I split my dataset again (60% training, 20% validating, 20% testing) and trained the new model only considering selected features. For this testing and validating datasets have been used. Then obtained accuracy score, classification report and confusion matrix for model evaluation. After this feature selection, model accuracy has been reduced to 95.61% which is little less than previous accuracy. | 6 | 20 minutes |
| 9 | Final model testing | y\_pred\_sel = model\_sel.predict(X\_test\_sel)  accuracy\_sel = accuracy\_score(y\_test\_sel, y\_pred\_sel)  print("Accuracy (Feature Selected Model):", accuracy\_sel)  print("Classification Report (Feature Selected Model):\n", classification\_report(y\_test\_sel, y\_pred\_sel))  print("Confusion Matrix (Feature Selected Model):\n", confusion\_matrix(y\_test\_sel, y\_pred\_sel)) | Finally, model was predicted using test dataset and accuracy score, classification report and confusion matrix was obtained again to check the final model performance. The accuracy was 94.7%. | 5 | 10 minutes |