Kwasi Mensah

IST 718

Big Data Analytics

Lab Exercise 3

**Objective:**

1) Obtain data and understand data structures and data elements.

2) Scrub data using scripting methods, to include debugging, for data manipulation in R and other tools.

3) Explore data using essential qualitative analysis techniques including descriptive statistics.

4) Model relationships between data using the appropriate analytical methodologies matched to

the information and the needs of clients and users.

5) Interpret the data, model, analysis, and findings. Communicate the results in a meaningful way.

**Introduction**:

The Fashion-MNIST data set found on Kaggle, is a dataset that contains images of clothes and accessories. It has two parts: a training set with 60,000 examples and a test set with 10,000 examples. Each image is 28 pixels wide and 28 pixels tall, and each image is also assigned a label that represents the type of clothing it shows, such as a shirt , shoes, bags, etc. The creation of Fashion-MNIST designed it to be used as a replacement for another dataset called MNIST. The MNIST dataset consists of many images of handwritten digits. These images were collected by a research institute, and just as in the Fashion-MNIST dataset, each image in the MNIST dataset is 28 pixels by 28 pixels as well.

The purpose of Fashion-MNIST is to provide a new dataset that can be used to test and compare different machine learning algorithms. It has the same size and organization as the original MNIST dataset, making it easy for us to use Fashion-MNIST as a substitute for MNIST when they want to evaluate different modeling techniques. In using these techniques for classifying the images we will look at the accuracy score of each model created, the time it took for each model training to complete, and the trade-offs of each approach used.

**Question:**

“Can we use algorithms and compute to identify clothing items? Specifically, can we determine which algorithm and compute methodology provides us the most efficient approach for classifying simple fashion images?”

**Data Sets Used for Analysis (Data Obtained):**

The data set was obtained from the Kaggle link provided. I decided to use the ‘idx’ files that were available to download. The Fashion-MNIST dataset of Zalando's article images obtained has a total of 60,000 training images and 10,000 testing images. Below are the variables for each of the training and testing images collected before any cleaning was performed.

*‘train\_images’ -* shape:(60000,28,28)

*‘test\_images’ -* shape:(10000,28,28)

Each training and test example image is assigned to one of the following labels, 0 - T-shirt/top, 1 - Trouser ,2 - Pullover, 3 - Dress, 4 - Coat, 5 - Sandal, 6 - Shirt ,7 - Sneaker ,8 - Bag , and 9 - Ankle boot. Below are the variables each of the training and testing labels collected:

*‘train\_labels’ - shape(60000,)*

*‘test\_labels’ - shape(10000,)*

**Data Cleaning (Scrub):**

First the training and testing set of images are reshaped into a flattened vector format for us to perform our analysis of our image classification. Because our image pixel dimensions are 28 by 28, we will multiply these dimensions and reshape the array collected for our training and testing sets into a 1-dimensional vector. The reshaped variables are below:

*‘x\_images’ – shape (60000, 784): This is our training dataset shape.*

*‘y\_images’ – shape (10000, 784): This is our testing dataset shape.*

Next, we normalized the testing and training sets by two different methods depending on the model used. For our neural networks created with the ‘TensorFlow’ package in python we used the function ‘tf.keras.utils.normalize’ to normalize our training and testing sets. For our multilayer perceptron, we normalized the training and testing sets by 255 because the pixels of each value have the darkness of the pixel ranging between 1 to 255. We normalize the dataset to scale the data into 0’s and 1’s, which make it easier for our neural networks to learn when we create these types of models.

**Explore:**

**A picture containing screenshot, text, cartoon, illustration

Description automatically generated**

Above we see a sample image of each type of clothing item in the dataset associated with each of their respective labels and id number. Below we how the items may vary in design by looking at 25 images from our training set collected on grayscale.

A picture containing clothing, dress, fashion, footwear

Description automatically generated

We will visualize the different types of images of shoes, sneakers, sandals, and ankle boots, to see if there could be any confusion when our model is attempting to predict the correct image label when testing.

A collage of different types of shoes

Description automatically generated with low confidence A collage of different styles of shoes

Description automatically generated with low confidence

A collage of different types of shoes

Description automatically generated with low confidence

We see that there are some similarities between the ankle boots available and the sneakers available when looking at our sample, and maybe issues with sandals and the other two types of shoes may come up as we see style 4 for sandals looks very similar in shape to both ankle boots and sneakers.

**Interpretation & Model:**

A total of seven models were created to perform our image classification. The methods decided on were the Gaussian Naive Bayes, Categorical Naive Bayes, Random Forest, K-Nearest Neighbors, Linear Support Vector Machine, Neural Networks with TensorFlow using Keras, and a Multilayer Perceptron model.

***Gaussian Naive Bayes***

Naive Bayes methods are a type of supervised learning algorithms that use Bayes' theorem with a simple assumption. The assumption is that each feature is independent of all other features, given the class variable. The ‘GaussianNB’ package in python implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian.

A picture containing text, screenshot, font, number

Description automatically generated

We see above in the classification report of our models when new data is introduced, that the label 9 (Ankle Boot) has the highest precision score of .9150, which tells us about the correctness of positive predictions. We see that label 7 (Sneakers) has the highest recall score of .9880, which focuses on the instances in the dataset that are true positive. Lastly, we see label 9( Ankle Boot) has the highest F1 Score of .7715, which is the harmonic(overall measure of performance) mean of precision and recall scores for each category .

***Categorical Naive Bayes***

The ‘CategoricalNB’ package in python is a Naive Bayes algorithm for working with categorical data. The Categorical Naive Bayes calculates the likelihood of each category in a feature based on how often that category appears within the samples of a particular class. It assumes that each feature has its own categorical distribution. During training, it estimates the probability of each category in a feature given a class by counting occurrences and applies a smoothing parameter to avoid division by zero.

A picture containing text, screenshot, font, number

Description automatically generated

We see above in the classification report of our models when new data is introduced, that the label 1 (Trouser) has the highest precision score of .9782 (Correct Prediction) We see that label 3 (Dress) has the highest recall score of .9810 (True Positive) Lastly, we see label 1( Trouser) has the highest F1 Score of .9816 (Harmonic Balance).

***Random Forest***

The ‘RandomForestClassifier’ package in python from scikit-learn is used as an implementation method of creating random forest algorithm for classification tasks. We decided on 100 estimators for our specific model to try to improve the accuracy of our model. The random forest algorithm combines multiple decision tree classifiers to make predictions. The model samples with replacement from the dataset, and randomly chooses a subset of our training dataset to train each tree. The algorithm then averages the predictions to avoid overfitting our model thus improving our accuracy as well.

A picture containing text, screenshot, font, number

Description automatically generated

We see above in the classification report of our models when new data is introduced, that the label 1 (Trouser) has the highest precision score of .9938 (Correct Prediction) We see that label 8 (Bag) has the highest recall score of .9720 (True Positive) Lastly, we see label 1( Trouser) has the highest F1 Score of .9761 (Harmonic Balance).

***K-Nearest Neighbors***

K-Nearest Neighbors algorithm does not try to build a single model, but instead it stores the data from our training set as individual instances. When classifying a new data point, it looks at the nearest neighbors of that point and assigns it the class that has the most representatives among the specified number of neighbors. The classification decision is made based on a majority vote from the nearby points. The ‘KNeighborsClassifier’ python package from scikit-learn library used to build the K-Nearest Neighbors (KNN) model. For our model we decided to create a model that had 3 neighbors to compare points with.

A picture containing text, screenshot, font, number

Description automatically generated

We see above in the classification report of our models when new data is introduced, that the label 5 (Sandals) has the highest precision score of .9929 (Correct Prediction) We see that label 9 (Ankle Boot) has the highest recall score of .9730 (True Positive) Lastly, we see label 1( Trouser) has the highest F1 Score of .9769 (Harmonic Balance).

***Linear Support Vector Machine***

We will use the ‘LinearSVC’ function in python to perform our Linear Support Vector Classifier. The Support Vector Machine is a supervised learning method used for classification as well as other means. The SVM has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme. In addition, they use a subset of training points in the decision function (called support vectors), so it is also memory efficient.

A screenshot of a computer

Description automatically generated with low confidence

We see above in the classification report of our models when new data is introduced, that the label 1 (Trouser) has the highest precision score of .9927 (Correct Prediction) We see that label 5 (Sandals) has the highest recall score of .9783 (True Positive) Lastly, we see label 1( Trouser) has the highest F1 Score of .9686 (Harmonic Balance).

***Neural Networks with TensorFlow using Keras***

The Keras Sequential model consists of three sets of convolution layers followed by max pooling layers. Each set contains a convolution layer (Conv2D) and a max pooling layer (MaxPooling2D). On top of these convolution blocks, there are four connected layers (Dense) with 128 units. The activation function used for this layer is the rectified linear unit (ReLU), denoted as 'relu'. Next, we add a dense layer with 10 units for our 10 categories. We add a layer to get the probability distribution over the 10 categories, using the softmax function to make sure that when all probabilities are added they equal 1. We can see this below in the model summary.

A screenshot of a computer

Description automatically generated with medium confidence

To find the accuracy and visualize the output we use the ‘Adam’ optimizer, which is commonly used, use the ‘ SparseCategoricalCrossentropy ‘ to get the Cross-entropy loss between the labels and predictions, with 10 epochs or passes over the training set. We use this Cross-entropy loss function when there are two or more label classes.

A picture containing text, line, screenshot, diagram

Description automatically generated

Above we see our two graphs for our training and validation loss & accuracy respectively. The graphs both show that around the 4th epochs that each the accuracy and loss compared to the training accuracy differs by a noticeable margin but achieves an overall accuracy of around 80% and a low loss value around 235.

Below we see the output of our predictions made by our Sequential model making its predictions based on which label had the highest probability score for each respective image in the testing set.

***A picture containing text, screenshot, diagram, rectangle

Description automatically generated*** ***A picture containing text, diagram, screenshot

Description automatically generated***

***Multilayer Perceptron***

The ‘NeuralNetMLP’ function created is used for the creation of multi-layer perceptron (MLP) classifier that makes use of neural networks. It is designed to learn from training data and make predictions. The neural network is designed to classify data into one our 10 labels, with the input shape being that of our training data. We make use of 100 hidden layers, because the more hidden layers we have the more accurate our model will be. In addition, we use 100 epochs, where our model learns from the data from back and forth passes, with a fast-learning rate, split the training data in groups of 50 to improve performance, and our training data is being shuffled for each pass of our specified number of epochs.

A picture containing text, screenshot, plot, line

Description automatically generated A picture containing text, screenshot, plot, line

Description automatically generated

To the left we see a rough plot of each of the 50 batch runs and to the right we plot the average cost for each batch, showing a smoother plot of the average cost over the epochs. We see in both graphs that the cost is decreasing with each increasing epoch, which tells us that the model is improving and learning to make better predictions over time. Below we see an image of the predicted labels for the images in our testing set that were misclassified. We see that the model has a hard time correctly classifying Pullovers and Coats maybe because they look like shirts.

A picture containing text, sketch, black and white, screenshot

Description automatically generated A picture containing text, screenshot, font

Description automatically generated

***Model Performance***

Below is the data frame created for our visualizations in which we ranked the seven models we created sorted by the accuracy of scores of our models when the testing data was introduced. We see that the Categorical Naïve Bayes, the Random Forest, and the Multilayer Perceptron were our top 3 models with the highest accuracy. A picture containing text, screenshot, font, number

Description automatically generated

Below we see the bar chart of the overall performance of our model’s compute times for each approach. We see that the Gaussian Naïve Bayes, Categorical Naïve Bayes, and our Keras Neural Network have the lowest compute times with around 1 second, 7 seconds, and 35 seconds respectively. We also see that out Linear SVM, KNN, and Random Forest took the most time to create and make predictions. Each took around 268 seconds, 115 seconds, and 98 seconds respectively.

A picture containing text, screenshot, diagram, number

Description automatically generated

Below we look at the accuracy scores of our models that were created using our training images and labels datasets. We see that Random Forest, KNN, and the MLP have the highest accuracy with around 99%, 92%, and 90% accuracy respectively. We see that our GNB, CNB, and Linear SVM have the lowest accuracy with around 59%, 79%, and 80% accuracy respectively.

A picture containing text, screenshot, number, software

Description automatically generated

Below we look at the accuracy scores of our models when new data is introduced to our model to make predictions. We see that Categorical Naïve Bayes, Random Forest, and the Multilayer Perceptron have the highest accuracy with around 90%, 88%, and 87% accuracy respectively. We see that our Keras Neural Network with TensorFlow, Linear SVM, and Gaussian Naïve Bayes have the lowest accuracy with around 83%, 77%, and 59% accuracy respectively.

A picture containing text, screenshot, colorfulness, number

Description automatically generated

**Conclusion:**

In terms of tradeoffs between each model created, for starters when we look at our Categorical Naïve Bayes, we see that it has a very low compute performance cost, but tested highest when new data was introduced to it. We could further improve on this model because its initial accuracy with the training data was not that high. When we look at the Random Forest we see a good balance overall and as a result first with overall performance. When we look at our MLP model we see that for both the training and testing sets it has a moderately high accuracy score, and a reasonable compute time. Although the KNN model did take some time to run the tradeoff of a high accuracy score when predicting image labels makes it a reliable model. We would want to look more into using Keras with TensorFlow and ways to improve it because we see that it has a very short compute time with moderately high accuracy scores, although not the top 4 we could maybe improve it if we add more layers but that may add more time.

We see that the models with the higher compute costs do not necessarily have a better accuracy score than expected. We see that the Random Forest has a good balance between the compute cost and the accuracy. The KNN and MLP show competitive performance in terms of accuracy scores and processing time, while we see that our Gaussian Naïve Bayes being the quickest in cost time, has the worst accuracy score for both the training and testing sets.

One change I would like to make use of is different images rather than the two categories of Shirts vs. T-Shirts/tops. The two are very similar in design and may be confusing for our model to accurately distinguish between. In addition, we would like to do additional models to compare the result and further improve on the new techniques learned such as MLP, KNN, and our Neural Networks using Keras and avoid models with high compute times such as the Linear Support Vector Classifier for example, which has the worst compute time and low accuracy scores. Specifically, regarding the Neural Networks created using the Keras TensorFlow library I would like to look more into Convolutional Neural Networks to see if this method will give us an accuracy score of over 90%.

**References**:

[How to effortlessly explore your IDX dataset | by Chayma Zatout | MLearning.ai Medium](https://medium.com/mlearning-ai/how-to-effortlessly-explore-your-idx-dataset-97753246031f)

[Convolutional Neural Network (CNN)  |  TensorFlow Core](https://www.tensorflow.org/tutorials/images/cnn)

[1.6. Nearest Neighbors — scikit-learn 1.2.2 documentation](https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification)

[1.9. Naive Bayes — scikit-learn 1.2.2 documentation](https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes)

[sklearn.ensemble.RandomForestClassifier — scikit-learn 1.2.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

[1.4. Support Vector Machines — scikit-learn 1.2.2 documentation](https://scikit-learn.org/stable/modules/svm.html#svm-classification)

[sklearn.svm.LinearSVC — scikit-learn 1.2.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC)

[sklearn.naive\_bayes.CategoricalNB — scikit-learn 1.2.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.CategoricalNB.html#sklearn.naive_bayes.CategoricalNB)

[Image classification  |  TensorFlow Core](https://www.tensorflow.org/tutorials/images/classification)

[GitHub - 2SUBDA/Breakouts at Week8](https://github.com/2SUBDA/Breakouts/tree/Week8)

[(330) Deep Learning with Python, TensorFlow, and Keras tutorial - YouTube](https://www.youtube.com/watch?v=wQ8BIBpya2k&t=822s)

[matplotlib.axes.Axes.annotate — Matplotlib 3.7.1 documentation](https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.annotate.html)