Mateusz Siwy 19.01.2023

# Fundamentals of Data Science Winter Project

*Sources:*

Hands-On Machine Learning with Scikit-Learn and TensorFlow by Geron Aurelien

https://corochann.com/mnist-dataset-introduction-532/

https://www.kaggle.com/datasets/zalando-research/fashionmnist?resource=download

https://github.com/sharmaroshan/MNIST-Using-K-means/blob/master/KMeans%20Clustering%20for%20Imagery%20Analysis%20(Jupyter%20Notebook).ipynb

https://medium.com/@joel_34096/k-means-clustering-for-image-classification-a648f28bdc47

This project's task was to explore different data science techniques, by performing Dimensionality Reduction, Classification and Clustering on a given data set. Throughout this project I have used algorithms such as: PCA, KMeans, KNN and more. The dataset that we will be working on is called Fashion Mnist Dataset. We will be teaching the algorithm how to recognize what clothing article is present on the Image. In this project, aside from trying to get the best accuracy possible, it is important to recognize the differences between various image recognition methods, how some algorithms perform far better than others. This project was meant to be done without Deep Learning, therefore algorithms like CNN were not considered.

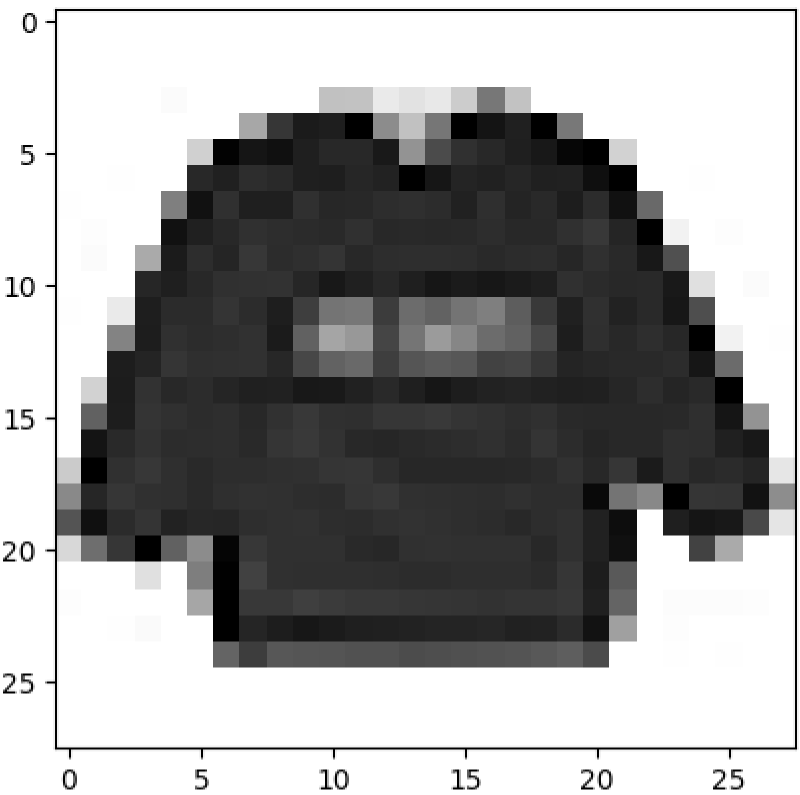
Firstly, we have to summarize the dataset. Fashion Mnist dataset consists of 70,000 labeled images of different clothing articles, each image is 28x28 pixels. The labels come in 10 classes as follows:

- 0 T-shirt/top
- 1 Trouser
- 2 Pullover
- 3 Dress
- 4 Coat
- 5 Sandal
- 6 Shirt
- 7 Sneaker
- 8 Bag
- 9 Ankle boot

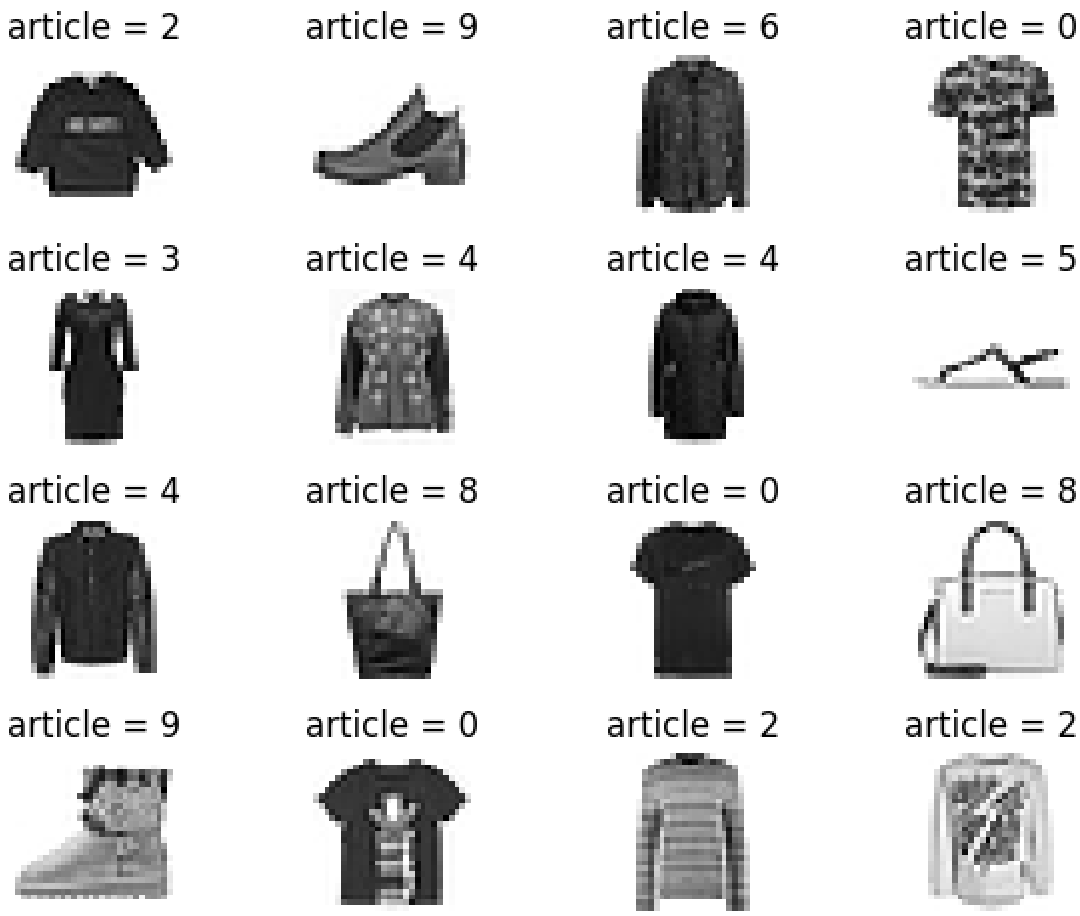
Each pixel has a single value associated with it (1-255), indicating the lightness and darkness of that pixel, with higher numbers meaning darker.

When working with labeled data, often supervised algorithms are what comes to mind first, but about that later. Between every classification task it is important to divide the data set into a train set and test set, to properly evaluate the algorithms performance in the end. Luckily, Fashion Mnist is already divided, first 60,000 records are for the train set, and the last 10,000 records are for test set.

Now that we know how our data set looks like, let's get to plotting some images.

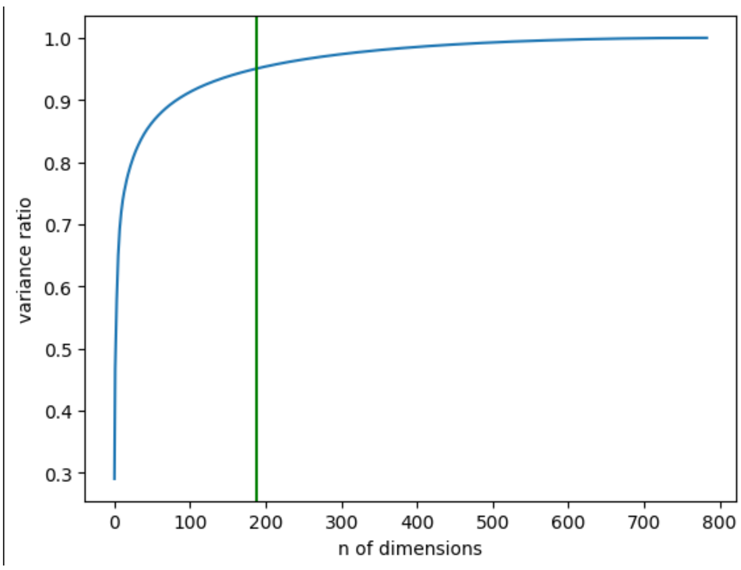



This is the first image in the data set, it's labeled with 2, which is a Pullover, the x and y axis contain pixels from the 28x28 format, this set has a shape of 60,000 x 784, because of that we have to reshape this row into 28x28 format. Now let's plot some more images to get a better feel at what we are working with.




Looking at this plot, we can already see which classes our algorithm might struggle with, for example articles 4 and 6 can be hard to distinguish even by human eye.

After understanding the data set, we can proceed to Dimensionality Reduction. It is an important step, because of the Fashion Mnist data set size. With 70,000 records, some algorithms will take a very long time to run. We will be using PCA Principal Component Analysis. To use it on our data set, we have to find the correct number of dimensions, we will be trying to use as little dimensions we can, while maintaining a strong variance ratio. I have decided to keep 95% of the original data.

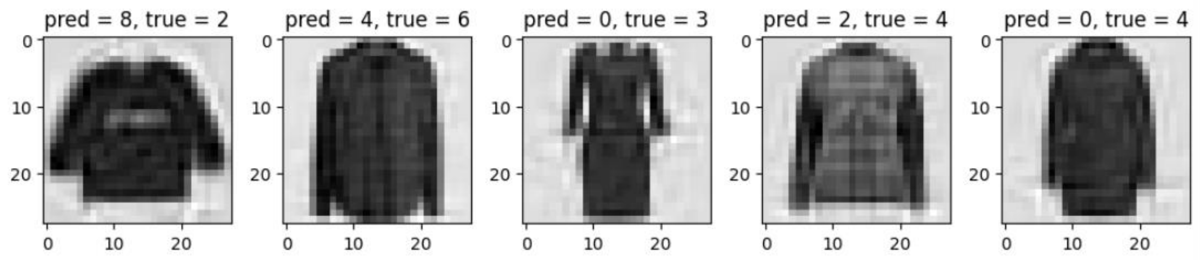


This plot shows the variance ratio to the number of dimensions, as we can see with about 187 dimensions, we maintain 95% of variance. We can also notice the plot's "elbow", it is a curve on a plot that often shows us the most cost-effective values. To make sure that we are not losing much information we can plot our train set after performing Principal Component Analysis.

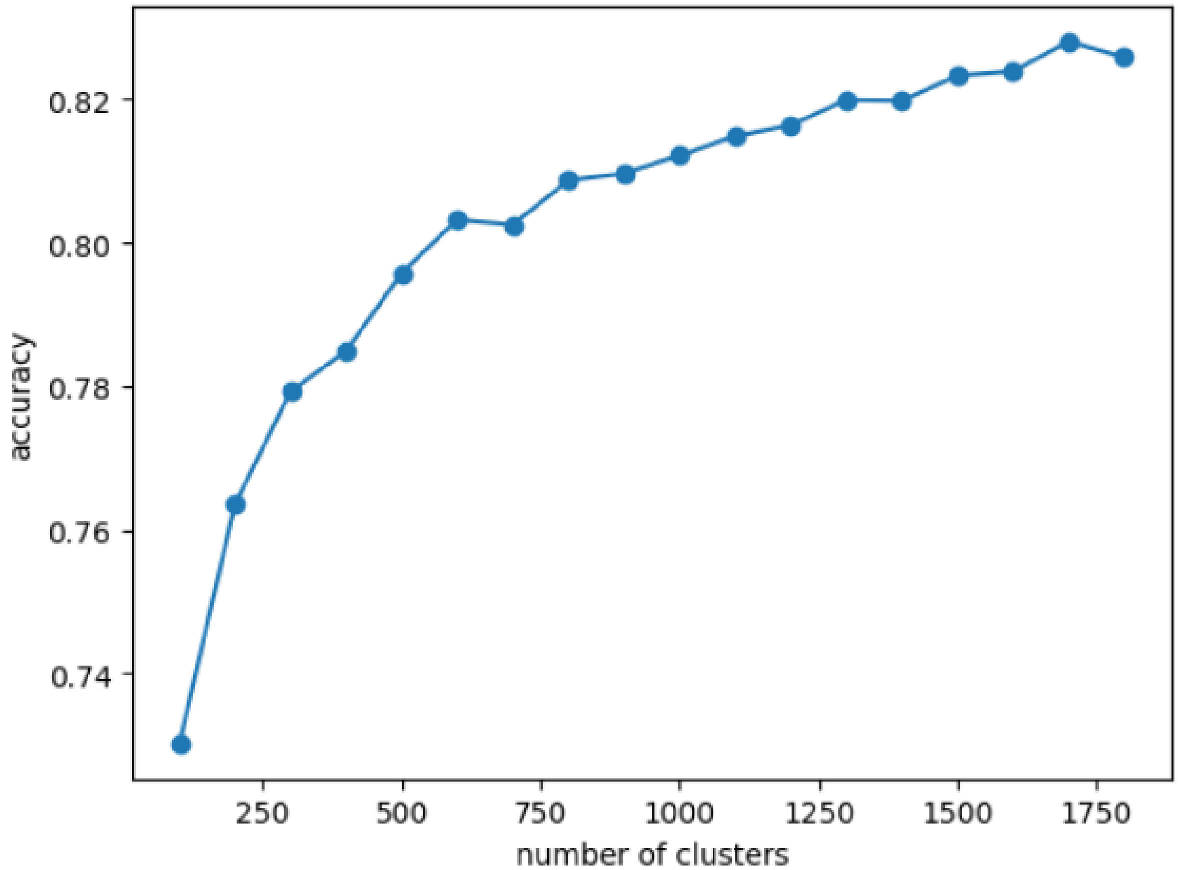

As we can see, almost no information was lost, this looks good.

Next step is to perform clustering on our data set, this is an unsupervised learning algorithm, so we will not be using our labels (y_train). I have decided to use Mini Batch version of KMeans, because of the size we are working with. Naturally, we will be setting the number of clusters to 10, we want every cluster to be equal to one of our original classes. Aside from size reduction, PCA also removed some of the noise from our images, which makes the clustering perform better.

Unfortunately, without proper optimazation, basic KMeans performed very poorly, it only got about 55% accuracy score. The easiest way to improve accuracy is to change the number of clusters, although 10 clusters seemed very logical. To get the correct amount we will be checking inertia and homogeneity of every number of clusters considered.




This plot shows which classes the KMeans algorithm has mistakenly assigned. For example, classes 2 and 4 (Pullover and Coat) were mistakenly clustered. To overcome this, we need to set the number of clusters to a higher number.




---

The previously described "elbow" is at about 80% accuracy with equals to ~500 clusters, which is 50 times more than the number we originally set.

To sum things up, KMeans clustering method performed really bad without proper optimization, after plotting accuracy against number of clusters we have established that the most profitable number is about 500 clusters, which is a lot than what we firstly set it to. Now let's compare our unsupervised learning algorithm to a supervised learning algorithm.

To properly choose the best classification for this algorithm, we will be using and comparing results of 3 different algorithms. To get a better feel of model's performance we will also be using cross_val_score, to divide our training set into folds, and perform the algorithm separately on each fold. The first one we are going to check is K Nearest Neighbors. To properly use KNN, we need to establish its hyperparameters, to do that we will use GridSearchCV. This method creates a grid for all hyperparameters under consideration and shows the best possible combination. After running this method a couple of times, the best number of neighbors seems to be 4, 6 or 8, all with similar accuracy at around 86%.

The second classification algorithm that we are going to use is called SGDClassifier, this is a linear algorithm, that relies on randomness during performing choices. Without any optimization the performance was at about 80%, which is still considerably high. This time, to optimize this algorithm we will be using the Standard Scaler method. After applying scaler on our train set, the accuracy improved to about 83%, which is a good improvement, but still worse than KNN. On the other hand, using Standard Scaler with KNN decreased the model's performance.

Lastly, we will be using Random Forest Classifier. This algorithm performed competitively well to KNN, it also got about 86% accuracy on test folds. In the end we will choose K Nearest Neighbor as our final classifier.

Finally, the accuracy score that we got on our test set is around 87%, I consider this accuracy to be really good, especially when done without any Deep Learning methods.

Now let's see what ChatGPT has to say about this task and compare its answers to our soluition.

# Chat GPT's answers

### My questions will be in normal font, and the chat gpt's will be bold. I have decided to shorten some of the answers, in order to make the report more concise.

- Do the summary of the data and discuss it.
- **The Fashion MNIST dataset is a popular image classification dataset with 60,000 training samples and 10,000 test samples. It consists of grayscale 28x28 pixel images representing 10 fashion categories, including T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. This dataset serves as a benchmark for evaluating machine learning models, offering a more challenging task than the original MNIST dataset due to the complexity of recognizing different fashion items. It is widely used for testing the robustness and generalization ability of image classification algorithms.**

This answer is unsurprisingly very similar to mine, this is something that can be read without any critical thinking and evaluation.

- Reduce data dimensionality
- **# Load the Fashion MNIST dataset**
- **(x_train, y_train), (_, _) = fashion_mnist.load_data()**
-
- **# Flatten the images into 1D arrays**
- **x_train_flat = x_train.reshape((x_train.shape[0], -1))**
-
- **# Standardize the data (optional but often recommended for PCA)**
- **x_train_standardized = (x_train_flat - np.mean(x_train_flat)) / np.std(x_train_flat)**
-
- **# Apply PCA for dimensionality reduction**
- **pca = PCA(n_components=2)**
- **x_pca = pca.fit_transform(x_train_standardized)**
-
- **# Apply t-SNE for dimensionality reduction**
- **tsne = TSNE(n_components=2, random_state=42)**
- **x_tsne = tsne.fit_transform(x_train_flat[:1000])  # Using a subset for faster processing**

As we can see, his answer is more or less correct, Chat GPT has the correct idea of what has to be done but fails to suit this answer to the dataset we are working on. When using PCA, Chat set the number of components to 2, therefore losing a lot of original data. It is important to note that Chat GPT can give us the correct answer to this problem, he just needs a lot of guiding to learn enough information to understand that PCA needs to keep ~95% of information.

- Visualize the reduced dataset
- **# Visualize the reduced dataset**
- **plt.figure(figsize=(10, 6))**
- **for i in range(10):**
-     **indices = y_train == i**
-     **plt.scatter(x_train_pca[indices, 0], x_train_pca[indices, 1], label=f'Class {i}', alpha=0.5)**
-
- **plt.title('Reduced Dataset Visualization using PCA')**
- **plt.xlabel('Principal Component 1')**
- **plt.ylabel('Principal Component 2')**
- **plt.legend()**
- **plt.show()**

Here Chat GPT plots given classes on a scatter, it uses 2D plot due to number of dimensions set in earlier answer.

- Cluster the dataset (and evaluate clustering results with classification labels).
- **# Apply K-means clustering**
- **n_clusters = 10  # Number of clusters (equal to the number of classes in Fashion MNIST)**
- **kmeans = KMeans(n_clusters=n_clusters, random_state=42)**
- **y_pred_train = kmeans.fit_predict(x_train_pca)**
- **y_pred_test = kmeans.predict(x_test_pca)**
-
- **# Visualize the clustering results using PCA**
- **plt.figure(figsize=(10, 6))**
- **for cluster_label in range(n_clusters):**
-    **cluster_indices = y_pred_train == cluster_label**
-    **plt.scatter(x_train_pca[cluster_indices, 0], x_train_pca[cluster_indices, 1], label=f'Cluster {cluster_label}', alpha=0.5)**
-
- **plt.title('Clustering Results using K-means and PCA')**
- **plt.xlabel('Principal Component 1')**
- **plt.ylabel('Principal Component 2')**
- **plt.legend()**
- **plt.show()**
-
- **# Evaluate clustering results using classification labels**
- **conf_matrix = confusion_matrix(y_train, y_pred_train)**
- **accuracy = accuracy_score(y_train, y_pred_train)**
-
- **print("Confusion Matrix:")**
- **print(conf_matrix)**
- **print("\nAccuracy:", accuracy)**

With this answer, Chat GPT assumed that the correct number of clusters would be 10 (the same as the number of classes in dataset), he fails to notice the user that number of clusters is something that alternate, leading to accuracy improvement. His evaluation of clustering is surprisingly good, he creates a confusion matrix which can become very useful during error checking.

We are going to skip the 'Split the dataset into training and testing.' section, because Chat GPT already did that in previous answers.

- **Perform classification and evaluate its result**
- **# Split the data into training and validation sets**
- **x_train_split, x_val_split, y_train_split, y_val_split = train_test_split(**
-    **x_train_pca, y_train, test_size=0.2, random_state=42**
- **)**
-
- **# Train a Support Vector Machine (SVM) classifier**
- **svm_classifier = SVC(kernel='linear', C=1)**

- **svm_classifier.fit(x_train_split, y_train_split)**
-
- **# Make predictions on the validation set**
- **y_pred_val = svm_classifier.predict(x_val_split)**
-
- **# Evaluate the classification results**
- **conf_matrix = confusion_matrix(y_val_split, y_pred_val)**
- **classification_rep = classification_report(y_val_split, y_pred_val)**
- **accuracy = accuracy_score(y_val_split, y_pred_val)**

Here, Chat GPT surprised me with his answer. Before classification, he split the data into training and validation sets. After that he trained a Support Vector Machine, which in the end gave really good accuracy. He even created a Confusion Matrix and Classification Report.

To sum things up, as we can see Chat GPT is a very powerful tool when used correctly. Without proper guiding he seems to oversimplificate the tasks that we are giving him. I also noticed that the more we converse with him, the more educated about the task he becomes.

Thank you for reading my report,

Mateusz Siwy