1. **About MNIST (digit view)**
2. **MNIST digit classification-Class\_Using NN (sklearn)**
3. **About SVM**
4. **SVM vs LogisticRegression**
5. **PCA vs t-SNE (T-Distributed Stochastic Neighbouring Entities**

**1. About MNIST (digit view)**

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.The database is also widely used for training and testing in the field of machine learning. The MNIST database contains 70,000 training images, which are handwritten digits and ere normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels. It consists of images digits like these:



It also includes labels for each image, telling us which digit it is. For example, the labels for the above images are 5, 0, 4, and 1. We can flatten this array into a vector of 28x28 = 784 numbers. It doesn't matter how we flatten the array, as long as we're consistent between images. From this perspective, the MNIST images are just a bunch of points in a 784-dimensional vector space.

Flattening the data throws away information about the 2D structure of the image.

***Isn't that bad…? Well, the best computer vision methods do exploit this structure, and we will in later labs.***

In this lab, we're going to train models to look at images and predict what digits they are. Our goal isn't to train a really elaborate model that achieves state-of-the-art performance. The simple methods we will be using here are MLP, softmax regression and SVM.

## 2. MLP Classifier in sklearn:

The scikit-learn in a scientific publication, initiated by ‘Google’ and later on received contribution from many different groups. Scikit-learn Machine Learning in Python is a Open-source, simple and efficient tools for data mining and data analysis, it is ccessible to everybody, and reusable in various contexts with built on NumPy, SciPy, and matplotlib, but commercially usable .

As a reference it is cited as: [Scikit-learn: Machine Learning in Python](http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html), Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

**MLPClassifier**(hidden\_layer\_sizes=(100, ), activation=’relu’, solver=’adam’, alpha=0.0001, batch\_size=’auto’, learning\_rate=’constant’, learning\_rate\_init=0.001, power\_t=0.5, max\_iter=200, shuffle=True, random\_state=None, tol=0.0001, verbose=False, warm\_start=False, momentum=0.9, nesterovs\_momentum=True, early\_stopping=False, validation\_fraction=0.1, beta\_1=0.9, beta\_2=0.999, epsilon=1e-08)

**activation** : {‘identity’, ‘logistic’, ‘tanh’, ‘relu’}, default ‘relu’

**solver** : {‘lbfgs’, ‘sgd’, ‘adam’}, default ‘adam’

The solver for weight optimization.

* ‘lbfgs’ is an optimizer in the family of quasi-Newton methods.
* ‘sgd’ refers to stochastic gradient descent.
* ‘adam’ refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba
* **Methods**

|  |  |
| --- | --- |
| [**fit**](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.fit)(X, y) | Fit the model to data matrix X and target(s) y. |
| [**get\_params**](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.get_params)([deep]) | Get parameters for this estimator. |
| [**predict**](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.predict)(X) | Predict using the multi-layer perceptron classifier |
| [**predict\_log\_proba**](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.predict_log_proba)(X) | Return the log of probability estimates. |
| [**predict\_proba**](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.predict_proba)(X) | Probability estimates. |
| [**score**](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.score)(X, y[, sample\_weight]) | Returns the mean accuracy on the given test data and labels. |
| [**set\_params**](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.set_params)(\*\*params) | Set the parameters of this estimator. |

## sklearn.linear\_model.LogisticRegression:

class sklearn.linear\_model.**LogisticRegression**(penalty=’l2’, dual=False, tol=0.0001, C=1.0, fit\_intercept=True, intercept\_scaling=1, class\_weight=None, random\_state=None, solver=’liblinear’, max\_iter=100, multi\_class=’ovr’, verbose=0, warm\_start=False, n\_jobs=1)

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the ‘multi\_class’ option is set to ‘ovr’, and uses the cross- entropy loss if the ‘multi\_class’ option is set to ‘multinomial’. (Currently the ‘multinomial’ option is supported only by the ‘lbfgs’, ‘sag’ and ‘newton-cg’ solvers.)

## Softmax Regressions

We know that every image in MNIST is of a handwritten digit between zero and nine. So there are only ten possible things that a given image can be. We want to be able to look at an image and give the probabilities for it being each digit. For example, our model might look at a picture of a nine and be 80% sure it's a nine, but give a 5% chance to it being an eight (because of the top loop) and a bit of probability to all the others because it isn't 100% sure.

This is a classic case where a softmax regression is a natural, simple model. If you want to assign probabilities to an object being one of several different things, softmax is the thing to do, because softmax gives us a list of values between 0 and 1 that add up to 1.

Support Vector Machine:

class sklearn.svm.**SVC**(C=1.0, kernel=’rbf’, degree=3, gamma=’auto’, coef0=0.0, shrinking=True, probability=False, tol=0.001, cache\_size=200, class\_weight=None, verbose=False, max\_iter=-1, decision\_function\_shape=’ovr’, random\_state=None)

**C** : float, optional (default=1.0), Penalty parameter C of the error term.

**kernel** : string, optional (default=’rbf’)

Specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n\_samples, n\_samples).

**degree** : int, optional (default=3). Degree of the polynomial kernel function (‘poly’). Ignored by all other kernels.

**gamma** : float, optional (default=’auto’), Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. If gamma is ‘auto’ then 1/n\_features will be used instead.

**Dimensionality reduction using PCA**

PCA is a technique for reducing the number of dimensions in a dataset whilst retaining most information. It is using the correlation between some dimensions and tries to provide a minimum number of variables that keeps the maximum amount of variation or information about how the original data is distributed. It does not do this using guesswork but using hard mathematics and it uses something known as the eigenvalues and eigenvectors of the data-matrix. These eigenvectors of the covariance matrix have the property that they point along the major directions of variation in the data. These are the directions of maximum variation in a dataset.

### T-Distributed Stochastic Neighbouring Entities (t-SNE)

t-Distributed Stochastic Neighbor Embedding ([t-SNE](http://lvdmaaten.github.io/tsne/)) is another technique for dimensionality reduction and is particularly well suited for the visualization of high-dimensional datasets. Contrary to PCA it is not a mathematical technique but a probablistic one. t-Distributed stochastic neighbor embedding (t-SNE) minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding.

**Ref: Visualizing Data using t-SNE, Laurens van der Maaten & Geoffrey Hinton, Journal of Machine Learning Research 9 (2008) 2579-2605 .**