Group 150 Report

Questions

1. Key strengths and weaknesses:

1.1. **Gaussian Naive Bayes**

+ A simple, yet effective classifier.
+ Doesn't require large amounts of training data.
+ The Gaussian distribution method solves the zero-probability problem with normal NB.
+/- It's naive assumption of independence between features could be considered a weakness, however it still achieves surprisingly good results on a wide variety of classification tasks.

1.2. **Decision Tree Learners**

+ Decision trees are often overlooked (in comparison to neural nets) but highly versatile.3
+ Inherently exploits the properties of a tree structure (lower usage costs).
+ Each result can be easily explained/interpreted by looking at the underlying decisions in the tree.
- Prone to overfitting
- Small changes in data can result in very different trees.
- Despite their resilience to un-normalized data, they don't tolerate incomplete data sets and create biases in unbalanced data.

1.3. **Nearest Neighbours**

+ Very simple to understand and implement.
+ Appropriate for classification tasks on datasets with irregular decision boundaries.
+ Relies only on its **k** parameter and its distance metric. Due to this, we suppose it's appropriate for classification tasks where some form of 'distance' is defined (e.g. word embeddings).
- Very computationally intensive on large datasets.
- Sensitive to all kinds of noisiness (less-meaningful parameters) and also unbalances in the data.

1.4. **Support Vector Machines**

+ Effective on high-dimensional data sets.
+ Memory efficient.
- Unsuitable for large datasets.
- Like the kNN, it performs badly on noisy data

1.5. **Logistic Regression**

+ Resource-efficient, easy to implement and train.
+ Doesn't require much data-tuning.
- Prone to overfitting
- Doesn't work on non-linear problems because of its linear decision surface.

2. Effect of hyper-parameters

2.1. Decision Tree Learners

max_depth:
This parameter defines the maximum depth of our decision tree. Increasing its value will always improve our accuracy during testing, however that can lead to overfitting and decreased accuracy on our testing set. Setting it None means that our tree will expand until no more information can be gained by splitting, or until all leaves contain less than *min_samples_leaf* (default 2) samples.

min_samples_leaf:
This parameter defines the minimum number of samples required per leaf. A split will only be considered if both the resulting nodes have at least that many samples. This contributes to smoothing of the model.

random_state:
Since the implementation uses randomness to decide on splits, we can use a seed to make it deterministic.

2.2. Nearest Neighbours

n_neighbours:
The number of neighbours to take into account, also known as "**k**". It's better if *k* is odd to prevent tied votes.

weights:
The distance metric used by the algorithm. This can be simply Euclidean distance or a custom metric created by you.

2.3. Support Vector Machines

C:
This parameter defines the strength of our regularization. It's inversely proportional. If it's too large this gives our model less freedom and leads to overfitting, while if it's small this results in too much freedom and lesser performance on the training set.

kernel:
Our choice of kernel should depend on the type of our training data. It's simply a

measure of similarity between data. For example, if we have a curved decision boundary, “poly” is appropriate.

random_state:
Since the implementation uses randomness to make probability estimations, we can use a seed to make it deterministic.

2.4. Logistic Regression

C:
Same as SVM.

penalty:
The penalty function changes the way we scale our regularization parameter (C) according to different properties of our dataset or model. For example, scaling it as our number of samples grow.

random_state:
Gives our optimizer (which relies on randomness) a random state to always produce the same outcome.

Us Census

Data Exploration

Question 1

Exploring our dataset we see that we have 11 features of which 3 are numerical and 8 non-numerical. We also inspect the types of the features and the detailed information about our numerical ones. Exploring the target variable - the salary we got to the conclusion that the class distribution is imbalanced. The data is not evenly split ~3/4 out of ~16000 samples are negative. Even if we predict all samples as the most frequent class we would get a decent accuracy rate, which does not make sense because our model is not learning anything. So classification accuracy would not be enough here. We are going to use both f1 score and accuracy for performance metric.

Question 2

We should not use race and gender in our machine learning algorithm. Using these features we may create a biased model which discriminates against some groups. These 2 features should not affect our model in order to make it fair for everyone. Given that we believe we should drop race and gender from our training data.

Data Preparations

Question 1

Inspecting our dataset for missing values we see that we have 4 columns that contain such values:

- Education-num (240) : We will fill them with the mean from the rest of the rows.
- Workclass (936) - Inspecting our dataset we see that in the cases where workclass is missing is a subset of the rows where occupation is missing
- Occupation (1181) - Most likely the user has no occupation and left it blank. We are not going to drop those rows and instead fill them with 'NA'
- Native-country (300) : Probably the user did not want to specify it, so we will fill it with unknown.

Question 2

Since our model doesn't understand the semantics behind text labels, we are going to encode our non-numerical features ("workclass", "education", "marital-status", "occupation", "relationship", "native-country") to a unique number. This is a standard approach for encoding text features.

Question 3

As previously discussed in the data exploration part question 2 - we are not going to use sex and race for our algorithm because we think that would create a biased model. Therefore we will drop these two columns.

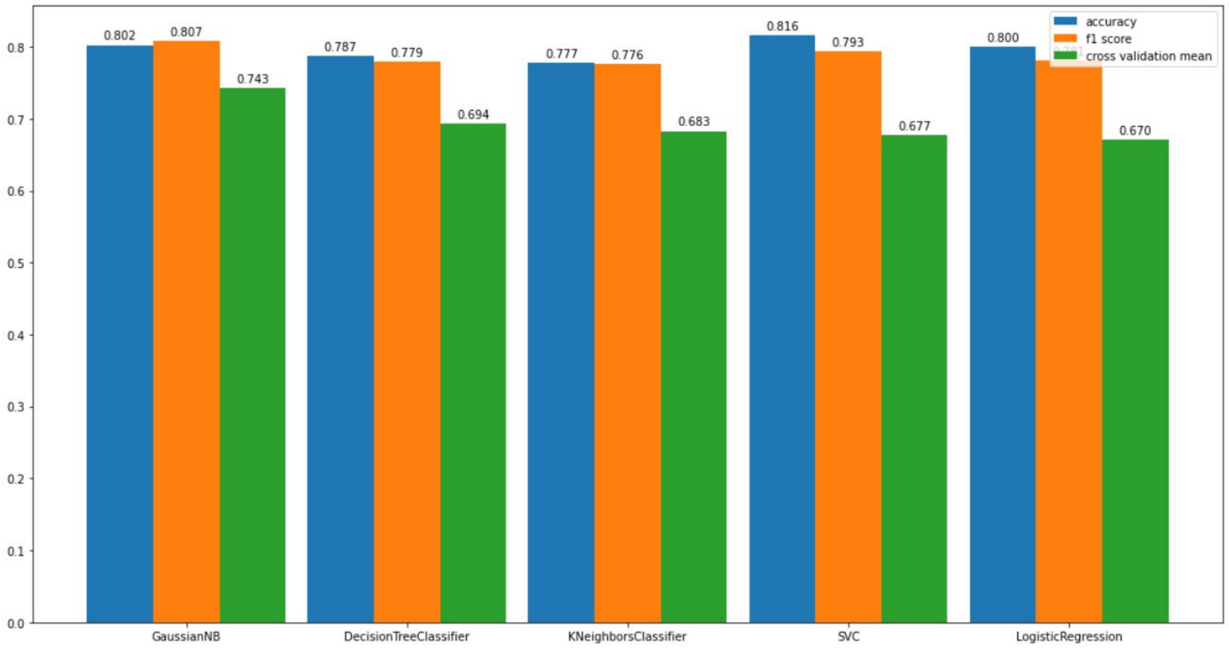
Experiments

Question 1

Splitting our dataset makes sure we do our training and validation on different data. Our performance is judged on performance on previously unseen data. We chose 70/30 for better generalization and lower training overhead.

Question 2

We made a plot that shows us the accuracy, f1 score and cross validation mean for each algorithm. Accuray would not be the best metric here because our target is not well balanced. GaussianNB is a too simple algorithm and it assumes that features are independent of each other. Surprisingly, decision trees do not perform so well given that we have nonlinear decision boundary and categorical data but maybe that is due to the default parameters which we will hypertune in the next part and most likely improve a lot. Logistic regression performs really well in accuracy and f1-score but its cross validation mean is the worst. However given that our dataset contains mostly categorical features it still not bad. KNN performs really well on our dataset given that we have mostly categorical features and KNN is not so good in this case.

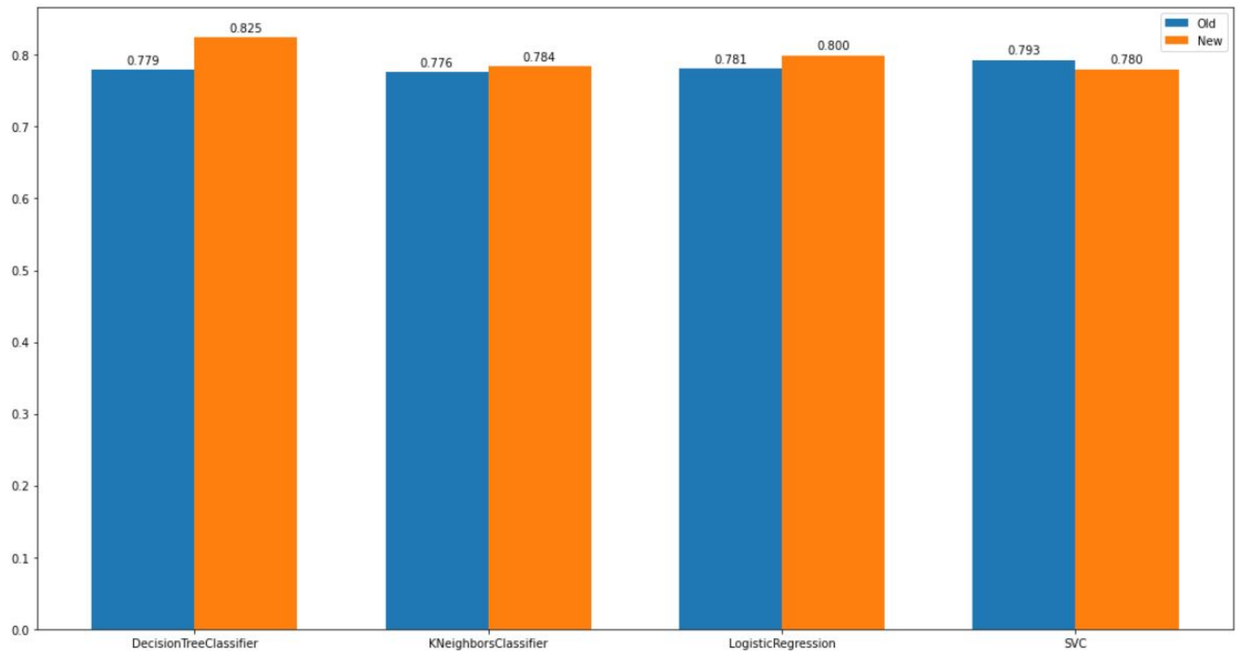


Question 3

For hyperparameter tuning, we'll use sklearn's GridSearchCV to find out the best hyperparameters for each model. Brute forcing all possible parameters is not practically possible so we had to narrow down the ranges of parameters that we are testing. We tried with a big range of parameters for each algorithm that we could computationally afford. For KNN we tried with a small number of neighbours(1-5) and with bigger values (20-25) and with both types of weights ["uniform", "distance"]. Based on our exploration we found that the best parameters are 26 neighbors with weights "uniform". Decision Trees were fastest to train and we could spend more time training them which is most probably the reason why it gave us the best score by improving the accuracy by 5%. SVC training was considerably slower than the others so we could not explore a wide range of parameters and we did not find better parameters than the default ones. Logistic Regression training was also quick so we tried a lot of different parameters despite this we could only improve our accuracy by 2%.

Question 4

The difference between the default parameter and our own surprisingly isn't that big except in the DecisionTree Classifier. We couldn’t get our SVC to improve any further in most of our parameter searches and we might even run into regressions if we touch the default values. KNN improves by quite a margin of <1% when using a bigger number of neighbours. LogisticRegression improves by quite a margin when using L2 penalty. The best parameter tuning was improving 5% the accuracy in the DecisionTree Classifier.

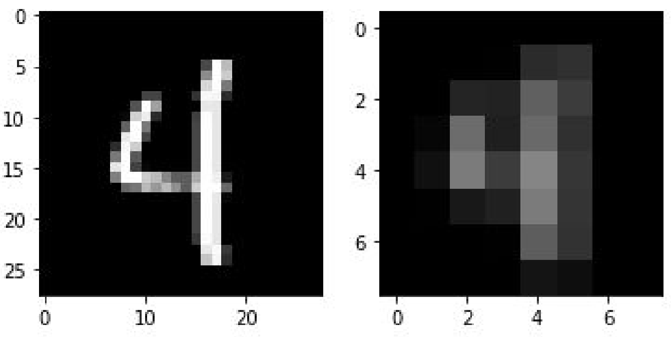


MNIST

Data Exploration

Question 1

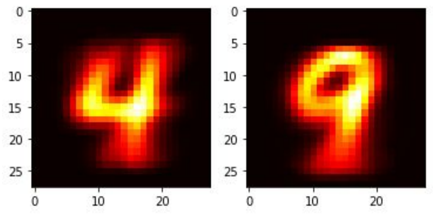
We can see there's some loss of data on the interpolated image. We lose fine details such as little squiggles and fine lines in the handwriting but that shouldn't impact our training much with methods we're using. However, the downsampled dataset will take a lot less time to train and evaluate. We're guessing the original data would be better for methods with deeper feature extraction such as deep neural nets.



Data Preparations

We reshape our data (flattening the 8x8 image to a 1x64 array). Our methods either require or perform better on processed data, e.g. standardized or clamped between [0,1] or [-1,+1]. Initially, we chose to scale it with sklearn's `preprocessing.MinMaxScaler()` so our X lies between [-1,+1]. However, normalizing gave us significantly better performance on our preferred methods. MNIST doesn't have any missing values so we don't need to worry about that.

We see that our dataset is quite balanced (there are no predominant labels). We can explore the dataset further, by plotting the heatmaps for the same classes. We can interpret 'heat' as correlation, which we'll use further down to reason about our classifiers.

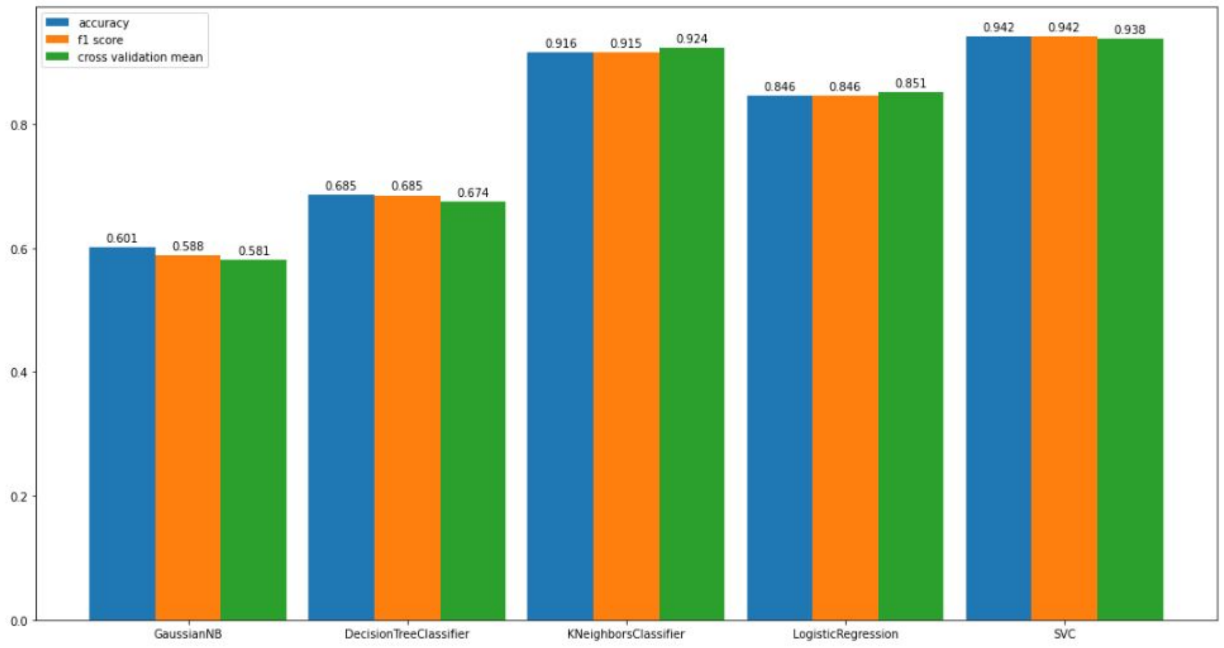


Experiments

Question 1

Splitting our dataset makes sure we do our training and validation on different data. Our performance is judged on performance on previously unseen data. We chose 50/50 for better generalization and lower training overhead. Our dataset is balanced so we can use a wide variety of performance metrics, including just plain accuracy.

Question 2



GaussianNB

NB's performance isn't surprising. We're dealing with heavily correlated data, as we've shown in our Data Exploration part. We could probably improve its performance with better pre-processing techniques and dimensionality reduction but we won't focus on that.

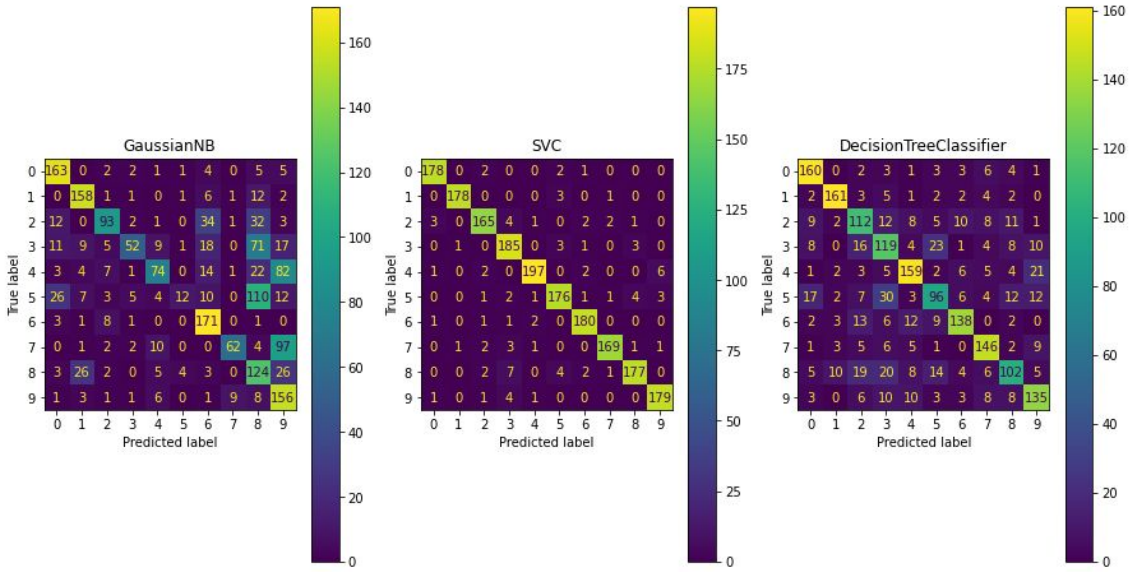
Decision trees

We also didn't expect much from the decision trees, however they're still doing surprisingly well on this type of data (just some ordered numbers and not categorical featues such as house_has_roof: true or cat_is_black: false).

KNN, SVM & Logistic Regression

As expected, the main contenders are the KNN classifier and the SVM. The models perform surprisingly well even with the default analysis.

We'll take a look at the confusion matrices of the most interesting models:



Our best models' matrices mostly look the same so we think this would be a more interesting comparison. We can observe some interesting things on GaussianNB's properties. Since it doesn't factor in correlated features, we can see a lot of missclassifcations due to a number having a part of another number. We can see it consistantly mistakes then number 5 4 for 9. In handwriting, a 5 and a 4 can be 'completed' to a 9.

On the other hand, the SVC performs quite well on all classes without differences too big.

Question 3

For hyperparmater tuning, we'll use sklearn's GridSearchCV to find out the best hyperparameters for each model. We're focusing on the prime 2 methods we think are suitable for the problem (KNN and SVC), and also LogisticRegression since it performed well with the default parameters. The parameters we settled on are in the global config above.

SVC

After being unable to get our SVC score any higher, we changed our test size and data preparation strategy. it seems like normalizing and using a 50/50 split gives the best performance. We can now go on to find out the best choice for kernel, C-value and other hyperparameterss.

Surprisingly, the poly kernel performs better in this configuration than rbf. We started our C value search in ranges around C=10, progressively going down lower and lower, which is a good thing for the generalization of our model. After ruling out RBF, we do more tuning. Our optimal C value lies somewhere around 1.0 and the optimal degree isn't far from the default 4. Searching for a better C value didn't make much sense, since the accuracy was pretty much the same for all values in the range (0.9; 1.1). This resulted in an f1-score of about 0.95 on the test set.

KNN

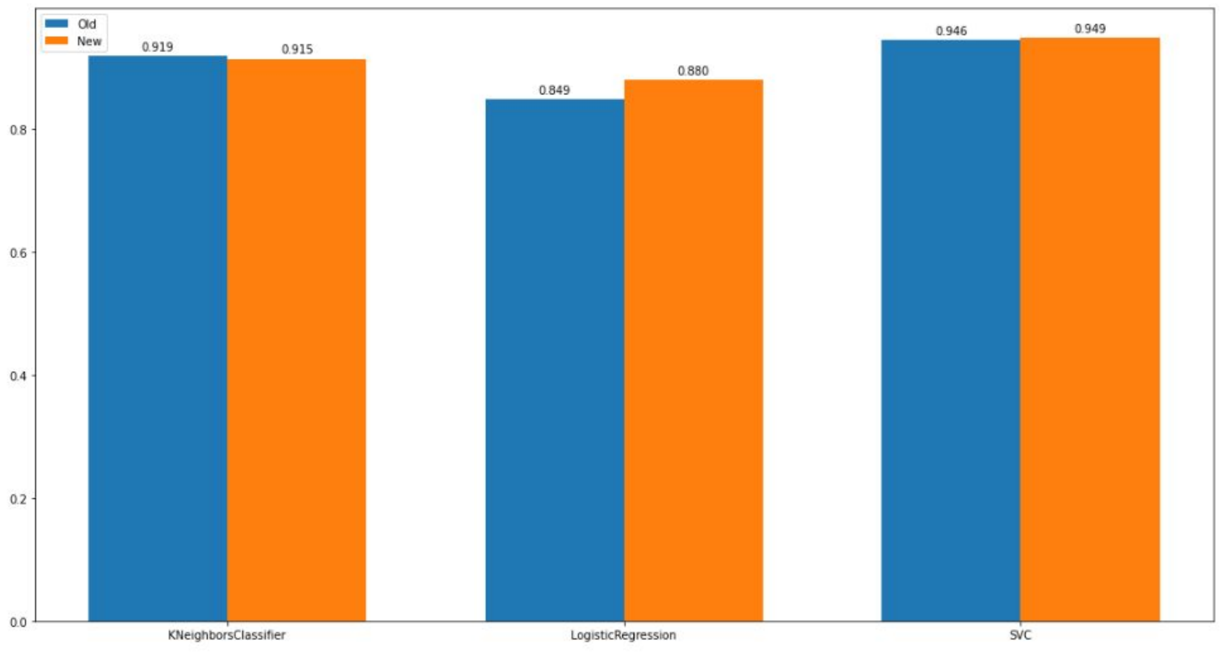
Right off the start, our most important parameter, at n_neighbors=4 or n_neighbors=3 seems to give us the best performance regardless of other settings. Changing our weights setting to distance gives us better performance too. Our naive interpretation is that by using the distance weight metric our classifier will be more sensitive to correlations in the data. The best result we've seen is a 0.94 f1-score on the test set.

Logistic Regression

A quick few runs and we can see that the default parameters are indeed one of the best. The only optimisation we can do is increase (decrease) our stopping criteria. This gives us an okay score of 0.89.

Question 4

The difference between the default parameter and our own surprisingly isn't that big. In fact, our choice of data preparation seemed to net us more accuracy in the end. Our cross-validation score is lower so we might've even overfitted a bit. We'll compare the f1-score of our main contenders:



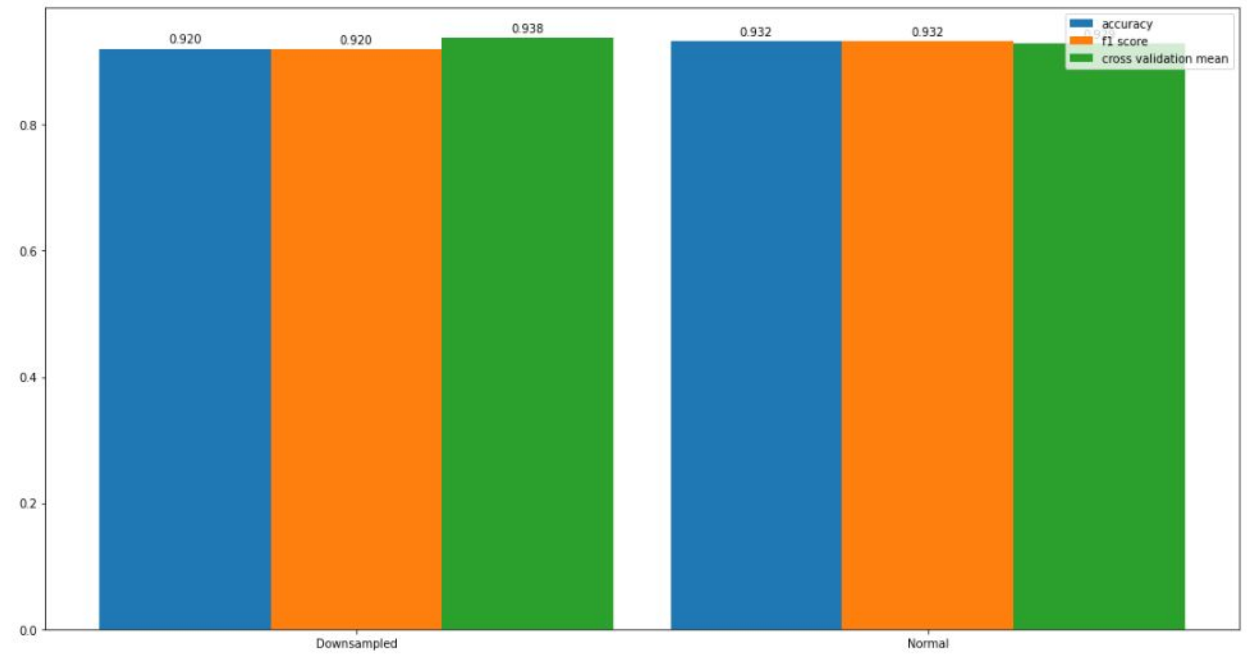
We couldn't get our KNN to improve any further in most of our parameter searches and we might even run into regressions if we touch the default values.

LogisticRegression improves by quite a margin when using L2 penalty and further tweaks to the C value and stopping criteria.

It was very hard to improve our SVC accuracy any further without additional pre-processing. However, its performance scores are still considerably very high so even slight improvements matter a lot.

Question 5

A conjecture is that some methods can perform better on the downsampled set than the normal one due to less noise in the data. Here we'll compare the SVC and KNN:



Depending on the results you see when running our notebook, our general opinion is that downsampling can indeed improve the accuracy of some of our models and we've seen it work in some configurations. However we have a gut feeling that using the normal dataset would give us a higher score on the hidden test set.

Conclusion

GaussianNB gave bad performance, as was expected and mentioned before. Both datasets were high-dimensional, complex and also heavily persistent on correlation between features(in both cases). NB just isn't fit for complex sets like these. In the case of MNIST, we noticed it making a lot of errors mistaking a number just because it looks like a ‘lesser’ part of another.

We expected decision trees to do well on the census and not so well on MNIST. It’s very hard to beat decision trees in the domain of decision problems like these, for example car/house price predictions, etc. On the other hand, just naively thinking, it would be a lot harder to classify features which are just plain pixel values in this way. If MNIST was a collection of more 'verbal' descriptions of digits (such as 'number of squiggles' or 'straight lines') it'd perform much better.

As a linear classifier we didn't expect Logistic Regression to perform well on both problems, but it showed some pretty good results. This is an incredibly simple model with very low overhead and surprisingly good performance that definitely shouldn't be overlooked. However we suspect it’s just blindly fitting on our data and would have problems on real generalization.

The KNN's performance is good as expected for problems such as MNIST. It could suffer a bit from too much data noise. A closer look at the MNIST dataset reveals some numbers are indeed very ugly or drawn off-axis and this could confuse the model. On the other hand we didn’t expect it to work at all for US Census, due to the lack of an accurate ‘distance’ metric that could be applied to the problem.

The SVC is best as expected on a multiclass problem with a small dataset. Like the KNN, it performs badly on noisy data and as demonstrated, it performs better on the de-noised downsampled 8x8 dataset with the same parameters.